NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

ANALYSIS FOR A TRUSTED COMPUTING BASE EXTENSION PROTOTYPE BOARD

by

Bora Turan

March 2000

Thesis Advisor: Cynthia E. Irvine
Second Reader: William A. Arbaugh

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 2000	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE Analysis For A TCBE Prototype Board	5. FUNDING NUMBERS
6. AUTHOR(S) Bora Turan	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
Approved for public release; distribution is unlimited	

13. ABSTRACT *(maximum 200 words)*

Agencies, institutions, individuals are demanding the use of commercial-off-the-shelf systems and cannot enforce mandatory security policies with these systems, which are equipped only with discretionary access controls. An inexpensive implementation of a multilevel secure local area network utilizing commercial-off-the-shelf hardware and software does not exist.

The The Naval Postgraduate School (NPS) is developing a Multilevel Secure Local Area Network (MLS LAN) to provide secure information sharing, classified at different security levels. The MLS LAN extends the high assurance of an evaluated multilevel secure system to a LAN that is formed by commercial personal computers (PCs) running commercial operating systems and office productivity software. The MLS LAN accomplishes the defined functionality by using custom boards which are designed to be plugged into personal computers. The boards are named the Trusted Computing Base Extension (TCBE). The TCBE is intended to provide trusted path and object reuse supporting services to the network TCB.

This thesis describes the hardware and software components, structures, interfaces required for the TCBE to complete a trusted path and control the client PC. Potential implementations are suggested and analyzed for security implications. A preliminary TCBE prototype has been constructed and tested for selected TCBE functions. It is shown that the TCBE prototype can be made both non-by-passable and tamper resistant.

SUBJECT TERMS Multilevel Security, Trusted Path, Hig h Assurance, Network Client			15. NUMBER OF PAGES 104
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

THIS PAGE INTENTIONALLY LEFT BLANK.

Approved for public release; distribution is unlimited

ANALYSIS FOR A TRUSTED COMPUTING BASE EXTENSION PROTOTYPE BOARD

Bora Turan
Lieutenant Junior Grade, Turkish Navy
B.S.E.E., Turkish Naval Academy, Tuzla Istanbul, 1994

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 2000

Author: _____
Bora Turan

Approved by: _____
Cynthia E. Irvine, Thesis Advisor

_____
William A. Arbaugh, Second Reader

_____
Jeffrey Knorr, Chairman
Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK.

ABSTRACT

Agencies, institutions, individuals are demanding the use of commercial-off-the-shelf systems and cannot enforce mandatory security policies with these systems, which are equipped only with discretionary access controls. An inexpensive implementation of a multilevel secure local area network utilizing commercial-off-the-shelf hardware and software does not exist.

The The Naval Postgraduate School (NPS) is developing a Multilevel Secure Local Area Network (MLS LAN) to provide secure information sharing, classified at different security levels. The MLS LAN extends the high assurance of an evaluated multilevel secure system to a LAN that is formed by commercial personal computers (PCs) running commercial operating systems and office productivity software. The MLS LAN accomplishes the defined functionality by using custom boards which are designed to be plugged into personal computers. The boards are named the Trusted Computing Base Extension (TCBE). The TCBE is intended to provide trusted path and object reuse supporting services to the network TCB.

This thesis describes the hardware and software components, structures, interfaces required for the TCBE to complete a trusted path and control the client PC. Potential implementations are suggested and analyzed for security implications. A preliminary TCBE prototype has been constructed and tested for selected TCBE functions. It is shown that the TCBE prototype can be made both non-by-passable and tamper resistant

THIS PAGE INTENTIONALLY LEFT BLANK.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENT

I am thankful to Turkey, my country, for making my degree and thesis possible. The support and encouragement given by my family made the experience one that I will never forget. Especially valuable were the input, support, and advice provided by my thesis advisor: Dr. Cynthia E. Irvine. Her knowledge and helpfulness in every aspect of the MLS LAN project provided the insight and motivation to seek out the best solutions every step of the way.

I dedicate my thesis to my wife Selin and to my son Kaan.

I. INTRODUCTION

A. BACKGROUND

Proper use, protection and handling of information are vital for every individual and every organization. Different sensitivity and criticality levels of information dictate different classifications and levels of control. While disclosure or modification of personal information may embarrass or be disruptive to an individual, exposure of critical national information may have serious consequences.

Every organization has a security policy; it is a collection of laws, rules, and procedures that reflect the security policy as carried out in its daily activities. Sterne [Ref. 1] defines organizational security policy as:

> The set of laws, rules, and practices that regulate how an organization manages, protects, and distributes resources to achieve specified security policy objectives.

Security policy is always intended to protect tangible assets [Ref. 1]. For every organization a security policy is needed dictating who has access to which asset. Different restrictions are applied to different types of information. For example, highly sensitive information can be available only to those having an appropriately high clearance.

Before the electronic storage and management of information, security policies were primarily directed towards the physical protection of tangible assets; for example, controlling the access to, limiting the duplication of and properly storing the protected document. The implementation of policies was straightforward, well established and well

understood. For example, most governments and military institutions had a security policy for handling sensitive material. Sensitive material was labeled with a classification level indicating who could access the information. Every person was given a clearance, which was generally associated with a job title, background information and that person's role. When a person wanted to access the information, the classification level on the information and the person's clearance were compared. If the clearance was higher than the classification level then the person was granted to access the information. There was a hierarchy for sharing information based on clearances and classifications. A person with a TOP-SECRET clearance could access TOP SECRET, SECRET and/or UNCLASSIFIED information, while a person with SECRET clearance could access SECRET and UNCLASSIFIED but not TOP SECRET information. These rules are still in force today: based upon a knowledge of another individual's clearance it is possible to determine whether or not classified information can be shared with that individual.

Security policies can be categorized into two classes: discretionary and nondiscretionary.

An access control policy is characterized as discretionary when a user may, at his or her own discretion, determine who is authorized to access the information he or she creates or manages. In other words, discretionary access rights limit access to the resources, but control over access rights is in the hands of users. For example, owner of a document decides who gets to see that document. Discretionary access controls are not necessarily persistent because the person who controls access to it, in effect controls the sensitivity level.

Nondiscretionary or mandatory access controls provide a fixed set of constraints on the access to information by the users. A mandatory access control policy reflects a set of rules for comparing access classes or classifications. Mandatory access controls are persistent, meaning that the sensivity of information stays the same at all times and global, meaning that information retains the same sensitivity wherever it is. Systems providing mandatory access control must assign sensitivity labels to all information resources (objects) and active system entities (subjects). An individual's sensitivity label specifies the level of trust associated with that person; and it is called a clearance. An object's sensitivity label specifies the level of trust that an individual must have to be able to access that information associated with the object. Mandatory access controls use these labels to determine who can access what information. Subjects acting on behalf of a user will have labels at or below the user's clearance.

In computers, security policy can be enforced by using discretionary or mandatory access controls depending on the type of policy being enforced.

In a computer system with discretionary access controls, selected users decide who can access the data and can specify the mode of access allowed. There are several mechanisms implemented on the computer systems for controlling discretionary access such as capability lists or access control lists. Although a discretionary access control policy is useful in some environments, it will not defend a system against malicious software such as Trojan horses. For example, a Trojan horse designed to make a copy of files that belong to a user can be embedded into a legitimate program. While the user is running the legitimate program, it runs the Trojan horse behind the scenes. The copy could be made readable to some other user who would not be intended to have access to the files. Discretionary access

controls do not offer adequate protection against malicious software and they are not sufficient to support multilevel security. Therefore, they are not adequate for requirements of military and government agencies that must enforce mandatory security policies.

In computers, mandatory access control policies are enforced by partitioning the information into sensitivity classes, and the authorizations of individuals, that is clearances, into access classes. Systems with mandatory access controls can manage multiple access classes simultaneously. They must assign sensitivity labels to all information (objects) and all individuals (subjects). Mandatory access control policies can protect assets from malicious software, if they are implemented with enough assurance.

Since the use of COTS software and any other software of unknown pedigree such as the programs downloaded from Internet have become popular, it is becoming much easier to implant malicious software. Therefore, it is crucial to guarantee the proper implementation and maintenance of mandatory access controls during the life cycle of software. For example, during distribution of the software or fixing of various problems.

Military and government agencies process classified information. Therfore, computer systems having mandatory access controls implemented with a high degree of assurance can ensure the protection of sensitive information while permitting the use of computers.

The enforcement of security policy requires that a set of functions or mechanisms be built into the computer systems. To enforce mandatory security policy, it is required that the needed security mechanisms be applied to all subjects and objects. Sterne [Ref. 1] describes both discretionary and nondiscretionary policies enforced by computers as automated security policy. He defines automated security policy as:

> The set of restrictions and properties that specify how a computing system prevents information and computing resources from being used to violate an organizational security policy.

For a mandatory policy, the mechanism must mediate every access. No subject may access an object directly. Instead, every subject must obtain access by going through the mechanism. By using a non-by-passable mechanism, it is guaranteed that every access is inspected by the mechanism and that the security policy is enforced.

The mechanism must be protected from modification. Since the mechanism is a a code executing on hardware, if it is possible to change the code, then it is possible to change the way that the mechanism behaves. Thus, the mechanism itself should be protected from illicit modification or tampering. This means that malicious code should not be able to engage in successful probing, penetration, or subversion attacks.

The mechanism should be verifiable as correct. Most software tends to be erroneous because it is usually written by programmers with marketing deadlines rather than being reliability and correctness in mind. As the number of lines of code and collaboration grows, the probability of having errors that are exploitable grows too. There are good software practices that can be used to address these problems but the bottom line is that the code that will enforce the mechanism should be small enough to be written without errors and be verifiable as correct.

What is described above mirrors the concept of reference monitor introduced by Anderson in 1972 [Ref. 2]. The reference monitor concept is implemented by a mechanism

in an operating system and it is described as a security kernel. The security kernel is concept developed by Schell [Ref. 3].

The TCSEC [Ref. 4] defines the security kernel as:

> The hardware, software, and firmware elements of a Trusted Computing Base that implement the reference monitor concept.

The Common Criteria's [Ref. 5] definition of the security kernel is:

> A set consisting of all hardware, software, and firmware of the TOE[1] that must be relied upon for the correct enforcement of the TSP[2].

Thus, to utilize the computers for processing and storing multilevel classified information, a mandatory security policy enforced by mandatory access controls should be implemented in a security kernel in the computer systems.

B. DELIVERING MULTIPLE LEVELS OF INFORMATION

The information that military and government agencies process is often classified. To provide access to information in a multilevel secure context, computers can be utilized in three modes of operation.

An easy and inexpensive approach is to physically place a separate single level **dedicated** computer at each desk for each classification level that the user needs to access. For example, a user with TOP SECRET clearance would need three computers. Since he has a TOP SECRET clearance, he can also read UNCLASSIFIED and SECRET information. Thus, he will have a TOP SECRET, a SECRET and an UNCLASSIFIED

---

[1] TOE represents target of evaluation in Common Criteria [Ref. 5]

[2] TSP are the TOE security policies in Common Criteria [Ref. 5]

computer. This mode brings some problems such as duplication of equipment, delayed information, inconsistent contents, and incoherent views.

Another mode is called **system high**, here all access permissions to information in the system will be set according to the highest classification level contained in the system and all data are restricted as if they were at the highest classification level. For example, a person lacking appropriate clearance could not access the system, even if the desired information had originally been at a lower classification and had been moved to the more highly classified system. In some ways, this implementation is worse than the first approach, since people don't really have any way of distinguishing information at different classification levels. The system high mode approach also introduces the problem of excessive clearances, and consequently, possible increasing laxity regarding classified information. Both approaches are unacceptable because without labels, one can not be sure whether information will be handled appropriately if converted to printed form, moved to another system or displayed.

Another solution was to place a multilevel secure system at each person's desk. A multilevel secure system can provide multilevel security by implementing labels and mandatory access controls. Such systems can differentiate between different security classes. Users can access the system at or below their clearance level. These systems mediate users' access to the information, thus enforcing the mandatory policy. The multilevel secure computer solution is perfect for providing information in multilevel secure context, but these systems are expensive, hard to use, and are not compatible with commercial-off-the-shelf (COTS) software. Incompatibility with commercial software is a big drawback, since a lot of software has already been written for commercial-off-the-shelf

platforms and users are accustomed to using it. Therefore, multilevel secure computer systems have never been widely used. To date, they are built in limited numbers, and are dedicated to specialized tasks.

C. MULTILEVEL SECURE LOCAL AREA NETWORK (MLS LAN) – A POSSIBLE SOLUTION

Even today, the situation has stayed the same: agencies, institutions, individuals are demanding the use of commercial-off-the-shelf systems and are trying to enforce mandatory security policy with these systems which are equipped only with discretionary access controls. An inexpensive implementation of a multilevel secure local area network utilizing commercial-off-the-shelf hardware and software does not exist.

An effective, practical MLS LAN implementation is needed that would utilize commercial-off-the-shelf hardware and software while enforcing a multilevel security policy. Any new MLS LAN should leverage existing computer technology including high assurance systems, COTS software, and COTS hardware. While multilevel secure high assurance systems are very expensive and limited in usability per desktop, they may be excellent choices for use as servers in a network environment. A practical MLS LAN solution would allow the use of COTS software running on COTS hardware as networked workstations to manage, retrieve and manipulate data that is stored on a high assurance multilevel server. A high assurance multilevel server could enforce mandatory security policy and could assure the security of the system by enforcing the integrity and the secrecy rules. Therefore managing the information in a multilevel secure context without compromising the security would be possible, if a network can be designed that has the ability to distribute information at multiple classification levels to COTS PC platforms. This

architecture eliminates single level redundant desktop computers; prevents inconsistency, and information delays; and supports rapid and cost-effective upgrades of software on client machines.

Our proposed solution is to equip client PCs with additional hardware, software components that will extend the assurance that the secure server provides. The extensions at the client workstations will provide the client PC the ability to retrieve and modify the information stored and controlled by high assurance server without subverting the security policy and therefore, will support security policy enforcement.

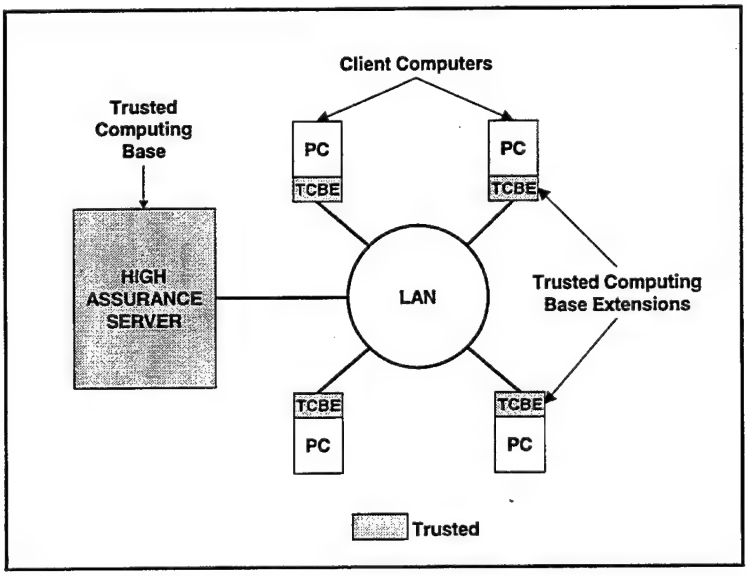

Figure 1. A Basic MLS LAN Architecture [Ref. 6]

A multi-level secure local area network (MLS-LAN) will consist of several components (see Figure 1).

A high-assurance multi-level trusted computer acts as a server. On this computer, an organization may store multiple levels of sensitive information with the assurance that the system will maintain the separation and integrity of the information with strict adherence to established MAC policy. In the MLS LAN project, the server will be Wang

XTS-300. The Wang XTS-300 is a high assurance multi-level secure computer that has been rated under TCSEC to Class B3.

Connected to this server, will be standard PC clients consisting of commercial-off-the-shelf (COTS) workstations running COTS operating systems and application software.

To extend the TC over the network, a component will be plugged into the PC client. It will reside as an intermediate layer between the XTS-300 and the client PC (Figure 2). It will support trusted computing base (TCB) functions of the MLS-LAN in the client PC as an extension of the TCB. It will recognize a secure attention key and will allow the establishment of a trusted path for identification and authentication and for negotiating a session level. It will provide mechanisms for secure I/O communication between the server and the client. It will also ensure proper object reuse at the client workstation, the purging of residual data from the previous session, which might be at a different session level. The component is called the trusted computing base extension (TCBE). The TCBE will be a custom add-on card plugged into each client PC.

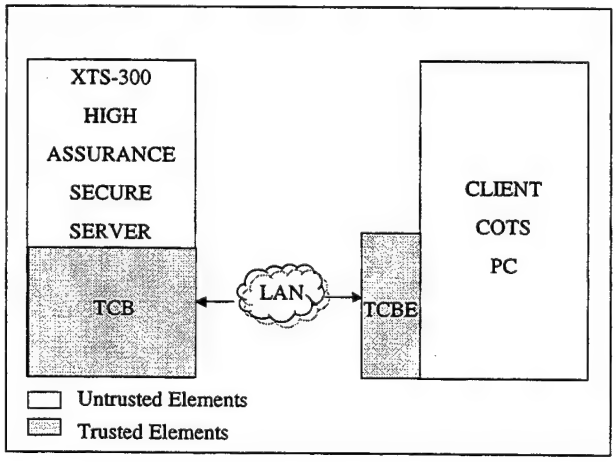

Figure 2. Trusted Computing Base Extension (TCBE)

D. GOALS AND THESIS QUESTIONS

The Naval Postgraduate School (NPS) Multilevel Secure Local Area Network (MLS LAN) project is building a client-server network and its objective is to enforce mandatory security policy while using the COTS software and hardware. It will provide multilevel information access to client COTS PCs through high assurance servers. The effort aims to provide affordable hardware and software without compromising security policy requirements.

The thesis will attempt to discuss requirements, design and implementation options for a TCBE built as an add-in card for the host COTS PC system. The design of the TCBE will be the primary focus of the thesis.

The requirements for the TCBE mandate its minimum configuration. It will be an add-in card (able to reside in client PC without modification of PC) that will have adequate computing power to control the operations to carry out the required functions such as communications setup, secure attention key (SAK) processing etc. It will have system's hard disk controller on board to guarantee the control of hard disk (main storage object). It will also have its own network interface in order to control the flow of information between the server and the host PC. The integration of existing hardware devices to build the TCBE will also be analyzed in this research.

E. ORGANIZATION OF THESIS

Chapter I has provided background information and an analysis of the problem addressed by the MLS LAN. It has discussed the need for a MLS LAN and described briefly, the NPS MLS LAN and its components. Chapter II discusses detailed analysis and

requirements for the TCBE. It will also discuss the functions of TCBE and will give an analysis of each function. Chapter III will address the required interfaces between the TCBE and the user, secure server, and host COTS PC. It will provide background information for interface technologies. Chapter IV will describe possible design topologies for the TCBE. It will give hardware details for implementation of different TCBE functions and will summarize a feasible solution. It will also present a security analysis of the candidate TCBE board. Then there will be a set of experiments presented which were designed to test some of the TCBE functions. Finally, Chapter V will summarize the thesis and will give the suggestions for the future work.

II. DETAILED ANALYSIS OF TCBE REQUIREMENTS

In this chapter, the concept of extending the trusted computing base will be presented. The chapter will define the requirements for the TCB extension (TCBE). According to the requirements and the functionalities needed, the components of the TCBE and where will they reside in the PC will be analyzed.

Finally, the chapter concludes with a TCBE design with its components, interfaces and design decisions.

A. WHY EXTEND THE TRUSTED COMPUTING BASE?

The concept of the trusted computing base (TCB) is fundamental to a trusted system. The TCSEC [Ref. 4] defines the TCB as:

> The totality of protection mechanisms within a computer system, including hardware, software, and firmware, the combination of which is responsible for enforcing a security policy. A TCB consists of one or more components that together enforce a unified security policy over a product or system. The ability of a TCB to enforce correctly a unified security policy depends solely on the mechanisms within the TCB and on the correct input by system administrative personnel of parameters (e.g., a user's clearance) related to the security policy.

The TCB is the mechanism that enforces security policy in a system. The boundary between the TCB and the other system components is called the Security Perimeter [Ref. 3]. In other words, security perimeter is the imaginary boundary between the security relevant and non-security relevant functions in the system. In the trusted systems, the TCB must be designed and implemented so that the security related components and functions are within the security perimeter.

A trusted path initiated by a secure attention key (SAK) provides a guaranteed and secure communication between the TCB and the user. It guarantees that the user is talking to the TCB and the TCB is talking to the user. To establish the trusted path, it must be ensured that both ends of communication cannot be spoofed and that all messages are tamperproof. This means that when the user initiates communications with the trusted computing base within the high assurance server, the user is assured to be communicating with the TCB. Vice a versa, the TCB must be guaranteed that it is communicating with the user, for example, the TCB may be a process executing on a known piece of equipment that can be uniquely and positively identified.

Since the trusted path is a prerequisite for a protected session, secure communication via trusted path is vital for the MLS LAN. Without establishing the trusted path, it is not possible to have an unambiguous user interface to the TCB. It cannot be guaranteed whether the user is actually communicating with the TCB or not. More crucially, it will not be possible to guarantee that any subsequent communications can be protected. The trusted path will be used to support user identification and authentication, negotiation of session levels, and invocation of trusted processes on the server side on behalf of the user.

Extending the trusted path within a physically unprotected LAN is one of the primary goals of this research.

Normally, when a local user wants to use a monolithic secure system, he or she has to login in to the system. To guarantee that the user is interacting with the TCB, a secure communication channel --a trusted path-- is established between the user and the TCB. A non-spoofable secure attention key stimulus unambiguously invokes the trusted path. By

using the trusted path, the user supplies identification information to the TCB and the TCB authenticates the user and a session starts on the system. Thus, a trusted path connects the user to the TCB.

In the distributed computing environments, logging in to a system remotely requires the ability to communicate with the remote system The TCB is extended by connecting the remote user to the TCB with an extended trusted path (see Figure 3). For example, an entire computer or even a local area network may be inside the security perimeter; the TCB and the distributed users are connected via trusted communication channels. The essential point here is to control communication between trusted and untrusted systems and networks.

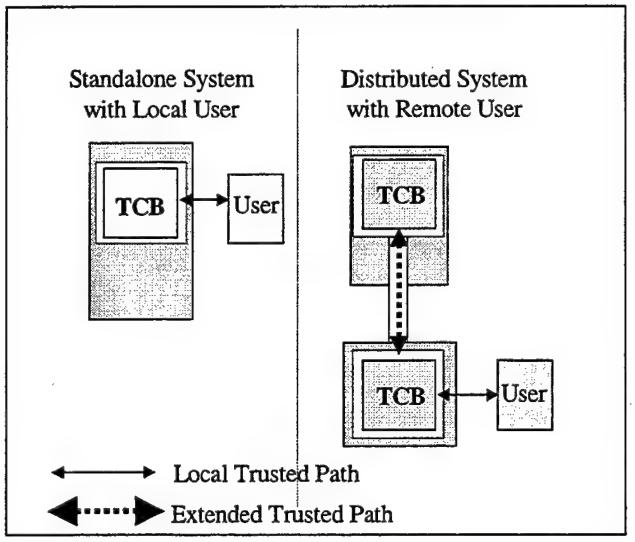

Figure 3. Extension of Trusted Computing Base

In the MLS LAN project, the TCB is extended through the network via a component called the trusted computing base extension (TCBE). The TCB residing inside the high assurance server, an XTS-300, is connected to the TCB running on the client PC via a trusted path as shown in Figure 4.

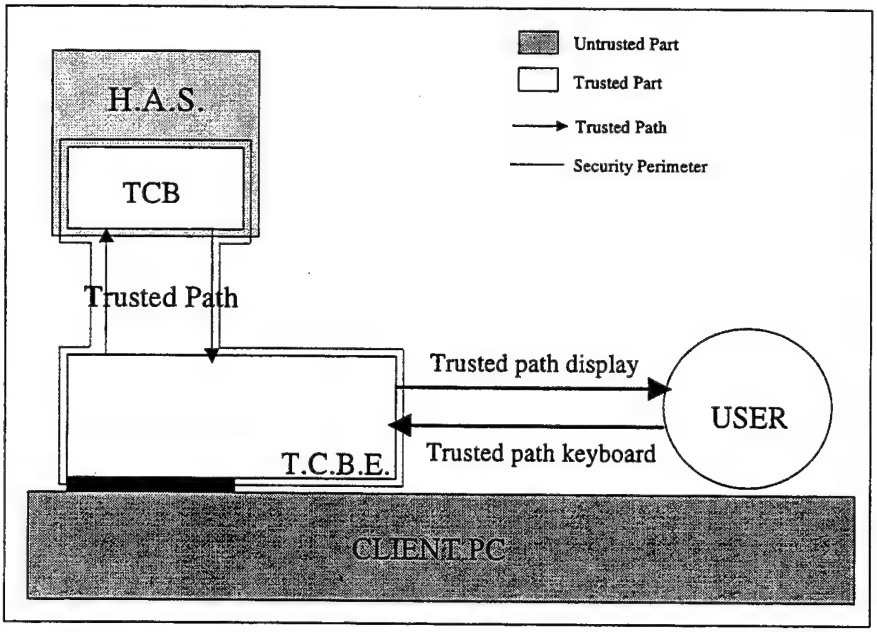

Figure 4. Extension of TCB through TCBE

B. REQUIREMENTS TO SUPPORT A DISTRIBUTED TRUSTED PATH

There is a set of steps that would be performed in sequence to properly distribute the trusted path:

First, the user stimulates the SAK sequence by pressing a key combination or a button. Then, the HAS and the TCBE create a protected communications tunnel. The tunnel will contain the trusted path; additionally, it might later contain other channels associated with the client (Figure 5). After successful establishment of the tunnel, the TCBE sends a protected token to the HAS to identify itself. Then the HAS responds with a prompt for identification and authentication. When created properly, the user is confident that the dialog he or she is having is with the TCB.

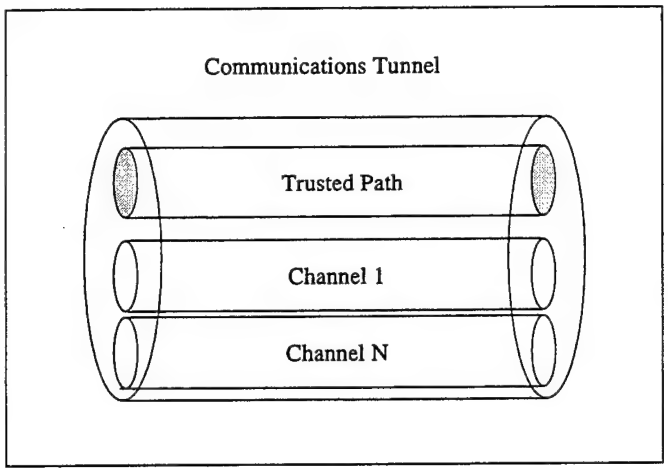

Figure 5. Communications Tunnel

Therefore, the first and the most important requirement for supporting a distributed trusted path is that the trusted path must assure the ability to identify and authenticate each candidate for creating a session on the high assurance server (HAS). A high assurance communications channel should be designed to authenticate the candidate client hardware to the server. First, it has to be ensured that both sides of the trusted path are connected to TCB components on the TCBE and the HAS. Then, there will be an authentication for the candidate user to the server, but it should be handled later within the process of the negotiation of the session level and other protocol parameters.

The second requirement is that the trusted path should be able to guarantee the integrity and the secrecy of the communications between the client and the server. Therefore, it must be assured that a third party identity can not read, manipulate or substitute the communications between the remote user and the server.

If these two requirements are met, the trusted path will provide hardware authentication and assurance that the communication has been established without disclosure, subversion and substitution of messages.

C. THE TCBE AS A SOLUTION

The NPS multilevel secure local area network is built upon the idea of extending the trusted computing base of a multilevel secure system to a network.

The Trusted Computing Base Extension (TCBE) will be a hardware and software component. It will be a separate computing device with a processor, memory and related circuitry. It will provide a computing platform to run the TCB code, while the host CPU will run untrusted code such as the Microsoft Windows NT operating system and COTS application software. Therefore, the TCBE will provide trusted support functions for the secure server's mandatory security policy and will permit remote identification and authentication of users, session level negotiation, downgrading information and other security-critical operations. The extension will provide client workstations with the capability to securely retrieve, manipulate and store the information in the server. The extension will also guarantee proper object reuse at the clients. Moreover, it will provide additional communication channels for application related communications between the server and the client PC. It will be an add-on card and it will be plugged into the client PC's PCI extension slot.

D. REQUIREMENTS FOR THE TCBE

There is a minimum set of requirements for the TCBE to carry out its support functions:

- The TCBE's support for the trusted path communications between high assurance server and itself

The existing mechanism for starting a session on the XTS-300 is to logon to the system by providing user identification and authentication information followed by the desired session level. To be able to start a session on the XTS-300 from a remote client PC, the remote user should communicate with the server's TCB and supply identification and authentication information. The TCBE will provide mechanisms to exchange hardware identification, authentication and user identification, and authentication with a desired session level by establishing a trusted path between the server and the client. The trusted path must be logically separate from the other client-server communications channels. It could be placed as a physical channel on a dedicated media or maintained as a logical channel multiplexed with other communications on the same physical media. For example, a logical channel can be created within a regular channel by encrypting the information which is intended for that logical channel. The TCBE must be able to provide mechanisms to protect information in the trusted path from disclosure and modification.

- Prevention of data remanence between sessions and proper object reuse

An operating system manages system resources and allocates them to itself and to applications. To effectively allocate limited physical resources, operating systems virtualize resources and multiplex physical resources to the virtual ones. In addition, when programs no longer need certain resources, the operating system switches the resources to another task or application. The most reissued resource in a typical system is generally the storage. By storage, we mean any storage location inside the computer such as memory, disk space, buffers, cache etc.

If any resource is not cleared of residual information before it is reissued to another subject, it may be possible for that subject directly or the programs running on behalf of that subject to read residual information. This is a major security concern.

A contemporary COTS PC system has enormous number of public and private storage locations. The motherboard, add-in cards, the video module and every piece of hardware is equipped with storage for performance, flexibility and expandability of the system. Contemporary COTS operating systems are designed with functionality, speed, ease of use and graphical user interface in mind without much concern for computer security. Therefore, operating systems often do not ensure the proper object reuse. In the MLS LAN project, a COTS operating system will be used as the client PC operating system. Thus, the TCBE has to make sure that every storage location inside the COTS client PC will be properly purged for every session prior to reuse. To accomplish this, the TCBE should be able to control every storage location to prevent unauthorized information leakage between sessions. Especially critical are the two main storage objects: the system memory and the hard disk space.

There are two possible ways to ensure secure reuse of storage objects. One is to overwrite (either by zeros or ones) the storage object before reissuing it to a new program. The other is to withhold the read access to the storage object until the object is write-committed by the new subject.

To ensure proper usage of memory objects, the TCBE must either:

- Have a hardware mechanism that has the ability to purge the memory used by previous session. It would be preferable to this without recycling the system

or

- Reset the host system before each session. This will purge the main system memory (RAM), which is non-volatile.

Agacayak [Ref. 8] provides an analysis of memory reuse along with possible solutions.

Operating systems use hard disk space to virtualize memory when there is not enough physical memory available. Operating systems swap memory contents in and out of hard disk space as they change between different tasks or processes. Therefore, it is also very important to ensure that residual sensitive information is not stored on the hard disk. To guarantee that hard disk storage objects are being properly reused, the TCBE will utilize a combination of read-only and read-write media. Balmer [Ref. 6] extensively analyzed delivery methods for operating systems by utilizing both read-only and read-write media. According to his work, for delivering the operating system to the TCBE; read-only media, either a CD-ROM or read-only hard drive, will be used to provide a fresh image of operating system. This approach will additionally guarantee the integrity of the client operating system, because the system will boot to a known state from a read only, non-modifiable media. The hard drives will be used interchangeably. One of them will be used for the current session and the other will be used for next session. Therefore, TCBE needs total control of the hard disk controller. This will be accomplished by removing the existing hard disk controller and adding a hard disk controller to the TCBE. This solution will guarantee the integrity of the operating system, will ensure proper object reuse, and will provide reasonable bandwidth to ghost the operating system or every session from read-only media to hard drives.

- Ability to respond properly to secure attention key (SAK) stimulus

Secure systems must provide an unspoofable hardware and software mechanism to initiate a trusted path between the TCB and the user. This can be used for security critical functions such as identification and authentication. This mechanism prevents malicious programs from impersonating the TCB and stealing user identification and authentication information or other security critical data. This mechanism is called secure attention key (SAK). The key is generally either a combination of keyboard strokes or a special button. The TCBE will provide a SAK. When it is invoked, the TCBE will communicate with the XTS-300 TCB by using a protected communication channel between itself and the HAS, and will provide a trusted path between the remote user at the PC and the TCB residing inside the XTS-300.

- The TCBE's control of information flow in and out of the client PC

Retrieval and the manipulation of application information will be handled by the untrusted client PC running untrusted code. Nevertheless, the ultimate storage of information will be done in the high assurance server. Thus, there will be a protected communication channel other than the trusted path between the server and the TCBE. The TCBE will need to control the session-related network communication from the client to the network and from the network to the client to prevent any information leakage. Since the retrieval and manipulation of the information will be done by the untrusted COTS operating system executing on COTS hardware, there will not be any way to prevent the distribution of sensitive information to the network, unless the TCBE controls the network interface. The TCBE must assure that there exists no other way for the information to move from the client to the network. Additionally for different states of the TCBE, such as SAK state or

the normal state, we need to be able to control the information flow between the server and the client. For example, when the SAK is pressed we want the TCBE to enter the a trusted path state and to halt any communications of client PC to/from the network; instead the TCBE provides non-by-passable network interface for COTS PC. It is always able to handle SAK processing. The TCBE will be the only interface to the network. This means that the TCBE will have the network interface card onboard. By routing the network traffic through TCBE, functionalities such as encryption can be added or modified without affecting the COTS PC hardware and software.

The features described here cannot be achieved only by rearranging COTS hardware and software. These requirements must be met through the design and construction of specialized software and hardware. Nevertheless, we aim to provide the additional regular functions without modifying the COTS PC hardware. Therefore, the TCBE will be an add-in card equipped with its own storage, storage controller (typically a hard disk controller), network interface card (an Ethernet NIC), and an embedded processor, which will provide the enough computing power to orchestrate the TCBE. Here, intend is not to evaluate a new computer and ideally a thin client can be used to simplify the TCBE architecture.

E. DESIGN AND IMPLEMENTATION OF TCBE

To this point, we have described the requirements for the TCBE. Now the challenge is to actually design and construct the TCBE. While designing the TCBE, there are two questions to be resolved:

- Where in the system will the TCBE reside?
- What peripherals are required on the TCBE?

F. THE TCBE LOCATION WITHIN THE SYSTEM

There are two ways to extend a COTS PC. The first way is to modify the motherboard and add new peripherals. This method requires very low-level design modifications, which actually means redesigning the motherboard. It is a very complex, erroneous, slow process and requires extensive experience and knowledge about PC architecture and peripherals. To implement the modification on dozens of computers, the same design and implementation effort would be consumed for each computer. Moreover, there are variety of motherboards and associated peripherals used in different brands and models of COTS PCs. The combinations are very numerous, so it is practically impossible to modify every PC to be extended. Therefore, motherboard modification is not a scalable solution.

The second method, which is the natural way of extending the PC architecture, is to design new peripherals as add-on cards. The extension cards should be designed to interface at the extension ports and they should physically fit one of the extension ports supplied with every computer. Therefore, the same add-on card can be used for every computer to be enhanced or extended. This method is easier, faster, less error prone, and most importantly scalable.

Nevertheless, it does not seem possible to be able to control the peripherals directly by designing the TCBE as an add-on card as proposed in the second approach. In Figure 6, a typical contemporary COTS PC architecture is shown. In the block diagram, it is clear that there is only one node that every peripheral in the system either directly or indirectly connected to. This node is called Host/PCI bridge and it connects the processor to every peripheral in the PC system. It is obvious that this node is the only place that can positively

control the communication between the CPU and the other devices, therefore this is the only place that can totally control the COTS PC system.

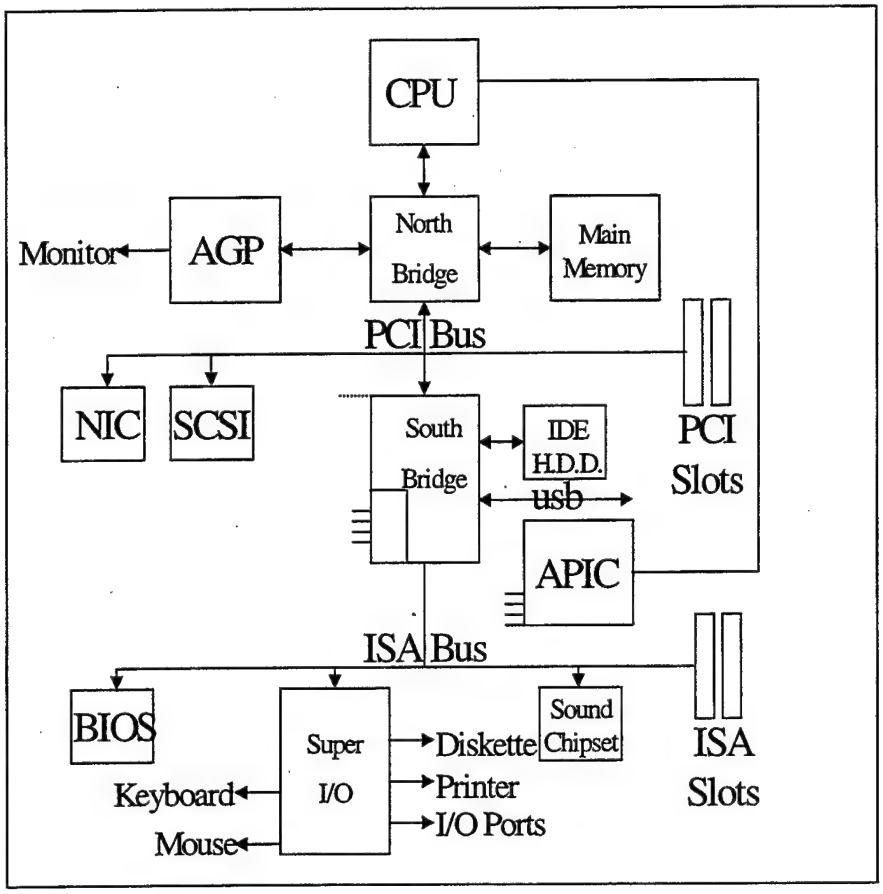

Figure 6. A Modern PC Architecture [Ref. 9]

The extension slots physically reside in such a location that they have no direct contact with most of the peripherals. Nevertheless, it is possible to design an add-on card as a PCI-to-PCI bridge with expansion slots on it. This solution can be a way of controlling the required peripherals by removing them from their original locations and placing them on to the add-on card's expansion slots. In other words, they are rerouted through the add-on card (see normal and rerouted operation in Figure 7). This method is not as complex as the previous approach, because there is an already a well-defined interface for expansion slots,

and all types of peripherals such as network interface cards and SCSI drive controllers are available commercially to be plugged into these slots. This solution is scalable, since once an add-on card is designed so that it can be used by any PC with the same expansion slot interface no matter what type of motherboard the PC has.

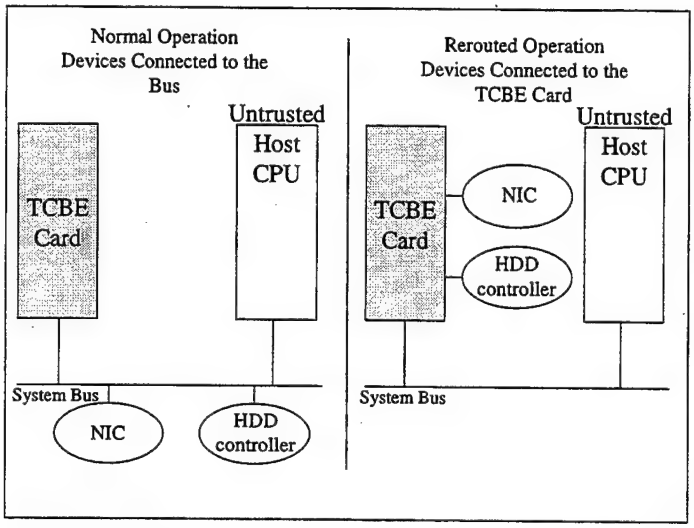

Figure 7. Control of Devices

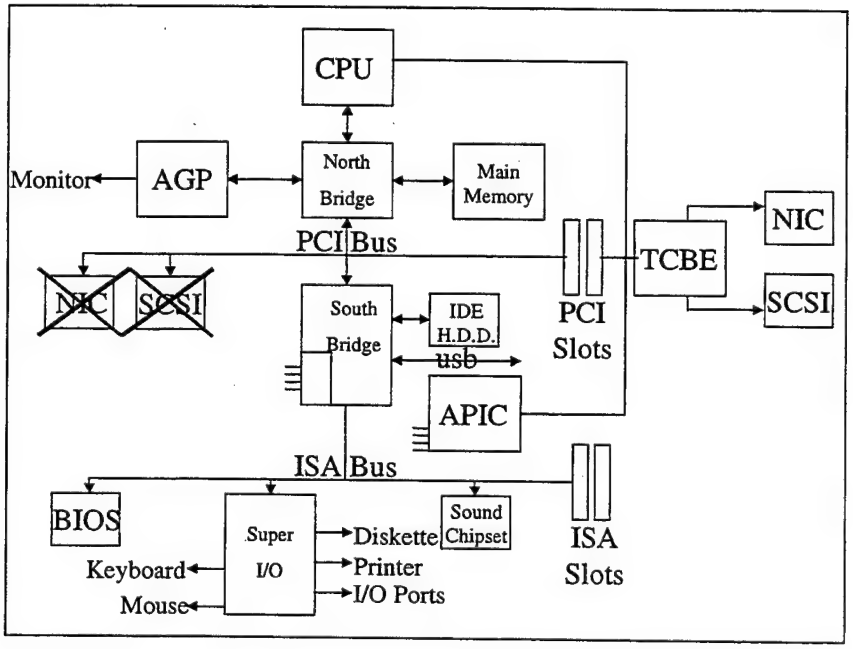

Figure 7a. Rerouted Peripherals

It is clear that design and construction of the TCBE, as an add-on card will be much easier, faster than the two alternatives. In addition, for the MLS LAN to be an affordable solution, we want an approach that will scale well; we want to build and integrate the TCBE without modifying the PC system. We will design and construct the TCBE as a PCI add-on card.

G. THE EXTENSION SLOT INTERFACE FOR THE TCBE

The decision to design the TCBE as an add-on card leads us to a discussion of the type of bus interface that the TCBE will be designed for. Modern PC architectures heavily use two types of bus architectures: one is the peripheral components interconnect (PCI) interface and the other is the industrial standard architecture (ISA) interface. Although almost every computer produced today uses the PCI architecture, there are still ISA peripherals and there are vendors that still produce and support ISA peripherals.

The original ISA architecture is a 16-bit architecture and it supports up to 8.33 Mbytes/sec transfer rates. For example, the keyboard and mouse controllers being used in today's computers are still ISA devices. During the evolution of PCs, ISA became Enhanced ISA (EISA), which is a 32-bit architecture. The EISA can transfer up to 33 Mbytes/sec. Although the EISA bus is 32 bits and provides transfer rates up to 33 Mbytes/sec, it still does not provide adequate bandwidth for today's 400 MHz processors. Therefore, EISA/ISA buses are not being used as the main system bus in modern PC architectures, but, as stated above, EISA is still found in the implementations of the side buses of a PC.

On the other hand, most of the components in today's computers and motherboards are PCI. The PCI architecture is a 32/64-bit architecture and it supports transfer rates up to 266 Mbytes/sec. Network interface cards, SCSI controllers, sound cards and most of the add-on cards today are being built as PCI devices. The PCI bus is today's main system bus since it can keep up with high processor frequencies.

In the following chapter, the interface of the TCBE to the host computer will be discussed and a detailed explanation of PCI bus architecture will be presented. Here we note that our choice of expansion bus interface for the TCBE will be PCI since it is a contemporary architecture, today's standard, and it can be easily assumed that it will be valid while the MLS LAN research is proceeding.

H. WHAT PERIPHERALS ARE REQUIRED ON THE TCBE?

As we stated above, a design that utilizes the expansion slots brings on the problem of peripheral control. The peripherals are normally under the host processor's control. To meet TCBE requirements, the TCBE must control some of the peripherals. The modern PC architecture (see Figure 6) is not designed to allow an add-on card to totally control the peripherals. An add-on card can only request control of a peripheral, that control may be granted by the host PC's processor, but this control is temporary. The host processor actually controls the peripherals on the system bus. Thus, the TCBE will need its own peripherals rather than those integrated on the system bus (see Figure 7a).

The requirements for the MLS LAN actually dictate the required peripherals on the TCBE:

- To be able to interface with the user, the TCBE should have trusted input and output onboard of its own,
- To interface with high assurance server and control the flow of information, a network adapter is needed,
- To control storage and enforce proper object reuse; it must support a hard disk controller,
- Finally, it needs a powerful enough processor with enough memory and accompanying components to orchestrate the TCBE.

To extend the TCB running in the HAS to the client PC, the user on the client PC has to be able to invoke the login procedure and supply identification and authentication information to the TCB, by using a trusted input and communication channel. Therefore, it must have user input interfaces. Balmer [Ref. 6] discussed different ways to provide user input and concluded that the TCBE must control its own keyboard. According to his analysis it will provide direct, dedicated connection between the TCBE and the user, eliminate the need to trust the host PC operating system to properly drive the keyboard and it will also provide an easier solution than modifying the PC BIOS to re-vector the keystrokes of the TCBE. Providing a keyboard means the TCBE must be equipped with related hardware circuitry, such as a keyboard controller and software programs, such as a TCBE keyboard driver, running on the TCBE's processor.

The system will be vulnerable to communication spoofing attacks without a protected display. Therefore, a display should also be incorporated. The problem arises if there is no private communication channel between the TCB and the user. A secure display can display TCB data to the user originating from the secure server via the TCBE. Balmer

[Ref. 6] discussed alternatives for the display of TCB information: a separate display or use of the PC's video display. He concluded that the TCBE must have its own protected display since it is not viable to totally control PC's display and have enough assurance that it is non-spoofable. Therefore, a display will be implemented as a part of the TCBE.

One of the design requirements for the TCBE is to ensure proper object reuse. The main permanent storage median for a PC system is its hard drive. This is very important for MLS LAN, since it is the media, from which the operating system is served to the client PC. The MLS LAN project will use a COTS operating system and in general, COTS operating systems are designed to be served from the hard drive. It is not possible to remove the requirement for the hard drive as the source of the operating system without modifying the operating system's source code. Moreover, COTS operating systems don't provide enough assurance that they will properly reuse the hard disk storage objects. It is therefore necessary to incorporate the hard disk controller on the TCBE to insure that the operating system is securely delivered to the PC for every session and that the hard disk is properly cleansed of sensitive information between every session. Balmer [Ref. 6] concluded his analysis of hard disk controller equipment on the TCBE by requiring that a hard disk controller interface be built into the TCBE. The reason behind is to have necessary controls for implementing the operating system delivery method, applying crypto (if needed) and ghosting to overwrite drives/partitions for object reuse purposes.

For the TCB to communicate with the TCBE, there must be channels between the TCB and the TCBE. These communication channels have to provide two kinds of communication: one for the trusted path and one for regular communication. These channels should run through the TCBE so that it will have the control of all

communications. Without the control of network communications it is not possible to guarantee against leakage of information to the network. Balmer [Ref. 6] concluded that the most reliable, secure way to provide the communication channels and to guarantee that the TCBE is the only conduit to the network, is to incorporate a network interface card onto the TCBE. The TCBE will be seen as a network interface card to the operating system and the network but it will be under total control of the TCBE and it will be the only path to the network. In addition to providing a gateway to the network, the TCBE can provide port remapping, since the system designed by Breyer-Joyner and Heller [Ref. 11] uses non-standard ports to communicate to the high assurance server (see Figure 8).

Therefore, we will integrate a standard network interface card onto the TCBE.

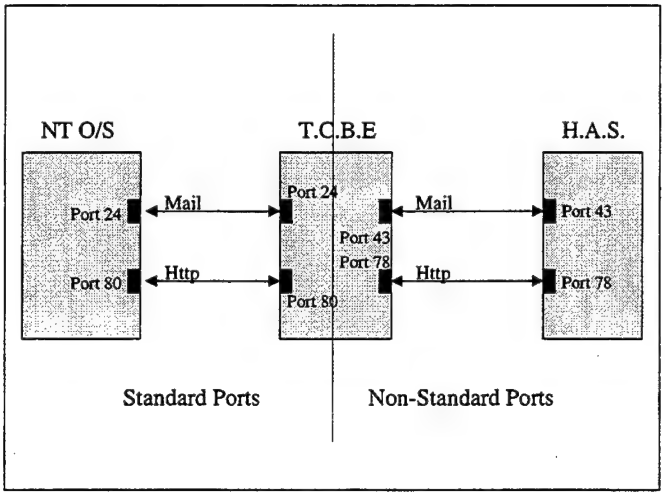

Figure 8. Port Remapping

To be able to manage the specified components and perform additional functions, an embedded processor will be chosen and added to the TCBE.

It is not trivial to design a PCI board from scratch. To avoid designing the entire TCBE from scratch, we will try to find readily available commercial-off-the-shelf hardware

modules to construct the TCBE and to make them work together. Therefore, the construction of the TCBE will be mostly an integration effort. This will, of course entail some hardware and significant software design.

To summarize the peripherals needed to satisfy the requirements, we will design the TCBE as a PCI add-on card, which will have its own keyboard, display, hard disk controller, network interface card and a processor.

To summarize the peripherals needed to satisfy the requirements, we will design the TCBE as a PCI add-on card, which will have its own keyboard, display, hard disk controller, and network interface card and a processor.

III. DETAILED ANALYSIS OF TCBE INTERFACES

In this chapter, the TCBE interfaces will be analyzed. Through this analysis, the underlying hardware technologies will be introduced. For each interface, the choices of the technology will be discussed and the final decision will be defined.

A. TCBE INTERFACES

The TCBE will provide the functionality needed to provide the support the TCB in the client PC. In addition to carrying out its tasks, it must be able to communicate with the host PC, the high assurance server and the user. The TCBE will be an add-on card with accompanying components. The TCBE will support multiple interfaces, which can be classified as internal and external interfaces:

- External Interfaces,
    1. TCBE interface presented to the user,
    2. TCBE interface presented to the host PC,
    3. TCBE interface presented to the high assurance server,
- Internal Interfaces,
    1. TCBE interface presented to the hard disk controller,
    2. TCBE interface presented to the network interface card,

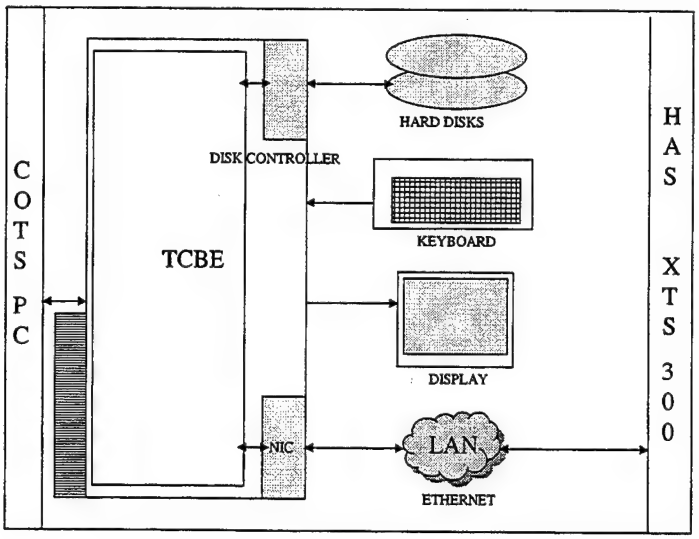

Figure 9. The TCBE Interfaces

Two interfaces need to be established between the user and the TCBE as input from the user and output to the user. The TCBE-to-host PC interface will be a PCI bus interface. This interface allows the TCBE to carry out required functions while residing on the system bus. The TCBE will also interface with XTS-300 by using an Ethernet connection and by running TCP/IP protocol. In the previous chapter, it was concluded that the TCBE would be constructed by integrating commercial-off-the-shelf components and by gluing them together with TCBE operating system and related software such as TCBE device drivers. There will be two internal interfaces within the TCBE: an interface to its hard disk controller, and that to a network interface card. In this chapter, interfaces and their underlying technologies will be analyzed.

B. EXTERNAL INTERFACES

1. TCBE Interface Presented to the User

As presented in the Chapter II and also by Balmer [Ref. 6], for the TCBE to securely communicate with the TCB, it is required to have a trusted keyboard and a protected display. The keyboard and the display will be integrated to the TCBE. In the following section, keyboard and display technologies and their internal operation are presented.

a. Keyboard

(1) Internal Operation. The information briefly provided below can be found in many computer hardware and architecture related resources such as [Ref. 9].

The keyboard and the associated keyboard interface are more complex than they seem. Every keyboard has a keyboard chip. The chip supervises a scan-matrix, which is formed using crossed electrical lines. A switch is located at each crossing. If a key is pressed then the switch is closed. The program in the keyboard chip detects keystrokes. The keyboard chip regularly checks the status of the scan matrix to determine the open or closed state of the switches. If a key is pressed, the code is determined by inspecting X and Y decoders and corresponding code is written into a keyboard buffer. Afterwards, keyboard sends the code as a serial data stream to the keyboard interface in the PC. Between the keyboard controller (on the PC) and the keyboard chip (on the keyboard case), there is a line for exchanging data and a line for keyboard clock. Therefore, the data are synchronous. The basic idea is that the keyboard chip sends the codes to the controller, upon receipt of codes, the controller issues a hardware interrupt via IRQ1 corresponding to

INT09H and provides the data at port B of 8255-Programmable Peripheral Interface (PPI) (Figure 11).

In keyboard systems, every key is assigned to a scan-code that identifies the key. Once the keyboard driver is in action (initialized), this scan-code value is converted into a ASCII character. Here is the detailed explanation of how the keyboard works:

When a key is pressed, keyboard generates a make-code (Figure 10), which is equal to the scan-code of the pressed key and transfers the make-code to PC's keyboard interface. There a hardware interrupt is issued and the interrupt handler fetches the make-code from the keyboard interface. The handler routine processes the code depending upon whether a function, a control key, or a normal key has been pressed.

When a pressed key is released then the keyboard again generates a code, called break code (Figure 10). The break code is treated using a procedure similar to that used for a make-code. The break code is simply the scan-code with the most significant bit is set. Therefore, the break-code is always equal to the make-code plus hundred and twenty eight. By loading the break-code, the handler can determine which key has been released and which key is still being pressed.

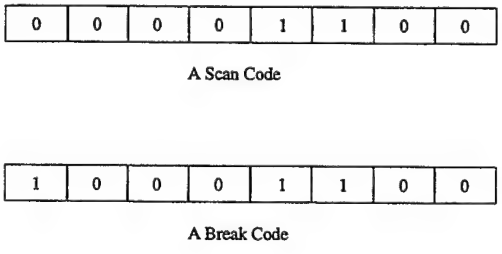

Figure 10. Scan and Break Codes

The interrupt handler passes the fetched codes to the keyboard driver. The keyboard driver uses an internal conversion table to assign the keyboard's scan-codes to ASCII codes. This allows a key to be assigned to a character or a function. By using another keyboard driver, the same computer can be used with a keyboard supporting another language such as a Turkish keyboard or a Chinese keyboard. The only thing that changes is that the new driver converts the scan-codes to another set of ASCII codes. In general, the keyboard driver can combine the two or more scan-codes to form key combinations such as CAPS + "a" which corresponds to "A".

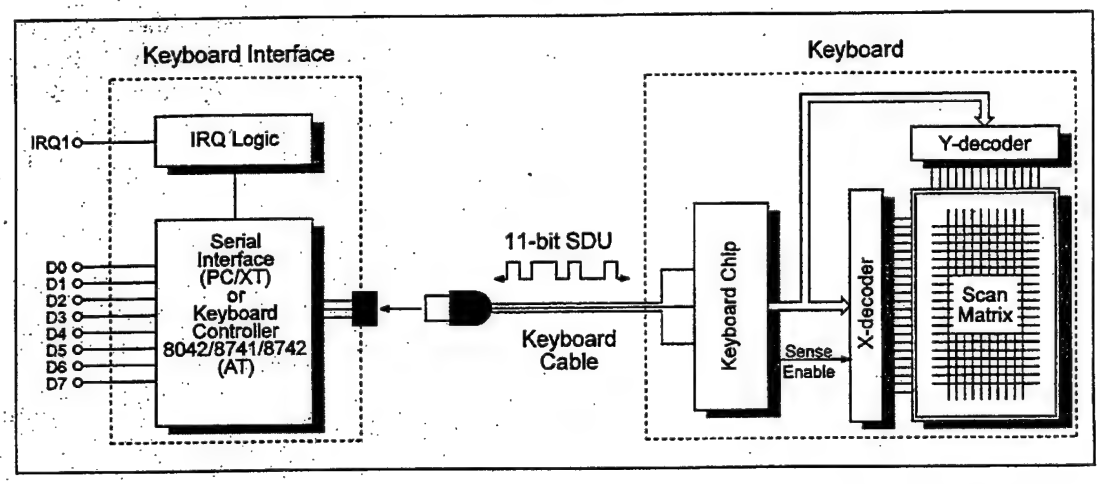

Figure 11. Keyboard Interface [Ref. 9]

(2) The TCBE Keyboard. The TCBE will incorporate its own keyboard. The function required from the TCBE is to supply a set of ASCII characters such as letters, numbers and special characters. In addition to that, the TCBE has to be able to sense the secure attention key (SAK) sequence and respond appropriately. The keyboard will be incorporated by adding a keyboard interface to the TCBE. The interface consists of IRQ logic and either a serial interface or a standard 8042/8741/8742 AT keyboard controller. The TCBE keyboard will be connected to the interface by a synchronous serial

line. The serial line will be composed of two lines: a data line and a clock line. Therefore, the TCBE must provide a clock signal for the keyboard. The TCBE operating system will need to run the TCBE keyboard driver. The driver will handle the regular keystrokes and convert scan codes coming from the keyboard to their corresponding ASCII characters. Moreover, the TCBE keyboard driver will continuously monitor key sequences and process the secure attention key (SAK) sequence. An internal TCBE interrupt will be assigned for keyboard interrupts. Whenever a key is pressed, the TCBE will issue a keyboard interrupt that will run the keyboard interrupt handler. The handler will pick the scan codes from the keyboard buffers and will pass the codes to the driver. The driver will process the codes and convert them to the corresponding ASCII values. If the SAK sequence is pressed, the TCBE will enter the SAK state and a special SAK handler will be started and communications between the TCBE and the TCB on the HAS will be established. A block diagram of TCBE keyboard is shown in Figure 12

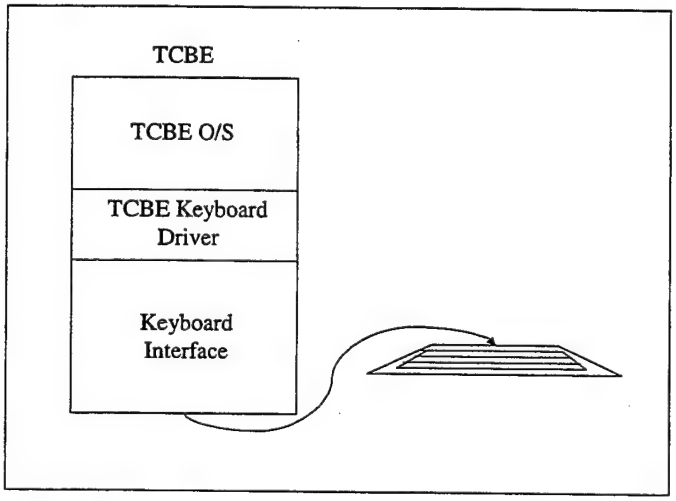

Figure 12. The TCBE Keyboard

b. The Display

(1) Internal Operation. The internal operation of display adapters is given by many general hardware resources as well as given by dedicated resources. The brief information given below can be found on any related resource (e.g. [Ref. 9]). The display of text and graphics on a cathode ray tube (CRT) computer monitor is similar to the method used in a TV set. An electron beam is guided vertically and horizontally to hit and stimulate the desired location on a fluorescent screen. The modulation of the electron beam has to be synchronized so that only the intended locations of the screen are illuminated. Scanning of the screen is carried out line by line; therefore, the image is displayed on the monitor in the same way. The graphics adapter has to provide the video signals required for the individual pixels as well as the synchronization signals for horizontal and vertical retraces. Liquid crystal displays (LCDs) and gas plasma monitors generate pictures in a manner similar to a CRT, but they don't have an electron beam forming the picture; instead, the individual pixels are assigned elements that may be addressed in succession. The image is also formed line by line.

In principal, graphic adapters don't differ significantly in structure. All the differences are associated with capabilities to display the number of colors and the resolution of the display.

The main element of a graphics adapter is the video controller or graphics control chip, generally called cathode ray tube controller (CRTC). The CRTC supervises the functions of the adapter and generates control signals. The host CPU writes information that identifies the text or the graphics to the video memory called video RAM

(vRAM). The CRTC continuously generates addresses for the video RAM to read corresponding characters and transfers them to the character generator.

In the text mode, the characters are usually defined by their ASCII values, which are further assigned an attribute. The attribute defines the display mode for the character. The character ROM holds a pixel pattern for every ASCII code. The character generator converts the character codes using the pixel pattern in character ROM into a bit sequence and transfers them into a shift register. The signal generator generates the necessary signals for monitor, using this bit stream from the shift register, the attribute from the video RAM and the synchronization signals from CRTC. The monitor processes these signals and displays the symbolic information in the video RAM. This modulates the electron beam of the monitor through the intermediate stages of character ROM, character generator, shift register and signal.

In the graphics mode, the information in the video RAM is directly used to generate characters. The entries in the RAM don't define an index into the character ROM, but already represent the pixel pattern themselves with related attributes. Therefore, no extra attribute information is required. The signal generator directly generates, from the bit values in the shift register, the brightness and color signals for the monitor. The same subsequent steps are carried out until the image is generated on the monitor.

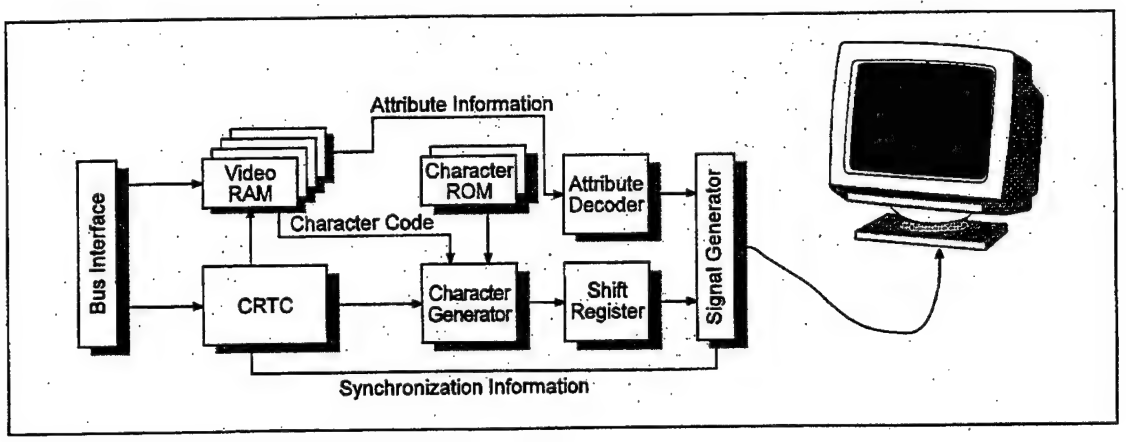

Figure 13. Display Interface [Ref. 9]

Modern adapters generally operate in two different modes: text mode and graphics mode. As explained above, text and graphics are generated differently so they require different video RAM size and organization. Generally graphics-intensive representations are much more resource intensive. On graphic adapters with video RAM up to 128 Kbytes, the whole display memory can be addressed via the CPU as a normal memory access. However, if the video RAM gets larger, then it would overlap the ROM extensions at address c0000H and would cause system wide problems. Therefore, adapters with more than 128 Kbytes of vRAM implement a switch that can be set by software to access various 128Kbyte windows in the video RAM.

As mentioned above, the CPU can access the video RAM using a basic memory cycle. Besides the CPU, the logic on the graphics adapter also performs accesses to the video RAM. Sometimes conflicts occur when the adapter and the CPU attempt to access the same address simultaneously. Modern adapters implement either a gate that blocks the CPU access while adapter is reading the RAM or a dual-port vRAM that has two data ports instead of a single switch data port.

Starting with the first PC, video adapters have evolved with the evolving demands. There have been a lot of different video adapters and adapter standards. Here is a brief overview of popular video adapters:

- Monochrome display adapter (MDA) was the very first PC video adapter standard. It can only be operated in the text mode. It is designed to be used with monochrome monitors. It offers 720X350 resolution and has 4 Kbytes of video RAM.
- Color graphics adapter (CGA) was the first graphics-capable, color adapter. It offers 640X200 resolution, can display 4 to 16 colors and has 16Kbytes of vRAM.
- The Enhanced graphics adapter (EGA) is the extended version of CGA. It is designed to overcome the low resolution and poor graphics of CGA. It offers 640X350 resolution, and can display from 16 to 64 colors. It has its own BIOS. It also allows the user to define his or her own characters and then load them to character ROM, so EGA is not limited to character defined by the BIOS.
- Video graphics adapter (VGA) provides high resolution (640X350) and better color (256 to 64000 colors). It can display 256 colors at a time. It has 256Kbytes of vRAM.
- The Super video graphics adapter (SVGA) is an advanced VGA. It offers 1024X768 pixels of resolution and 256 to 64000 colors. It has 1Mbytes of video RAM.

(2) The TCBE Display. The current method of communication from the TCB of the XTS-300 is text based. Therefore, to communicate from the TCBE, we will need a display, which is only required to present text used in the trusted path interface to the user. The simplest monochrome display will satisfy display requirements. With its small dimensions and availability, a LCD monochrome display will be suitable for TCBE display needs. The display will be connected to the TCBE via a LCD interface. A basic LCD interface consists of buffers and LCD driver circuits. A display driver will be written and will be a part of the TCBE operating system. The driver will send the text to the display interface where the characters will be processed and converted into character pixels to be displayed when the TCBE is in active communications setup phase. A block diagram of TCBE display is presented in Figure 14.

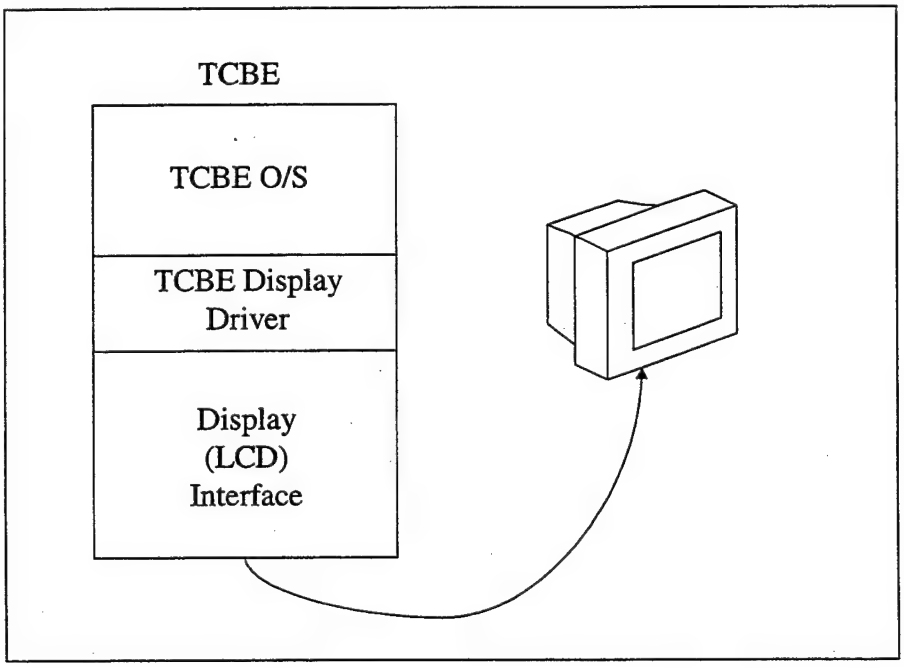

Figure 14. The TCBE Display

2. TCBE Interface Presented to the Host Client PC

In the following section, a brief summary of technical information about the PCI architecture will be given and then PCI issues related to the TCBE will be described.

a. PCI Architecture

Peripheral Component Interconnect (PCI) is a standard defined by Intel to prevent PC markets from becoming crowded with various permutations of local bus architectures. The PCI bus can be populated with adapters requiring fast accesses to each other and/or system memory. The speed of PCI comes from the ability of PCI to perform all read and write transactions as *burst transfers.* A burst transfer is one that consists of a single address phase followed by multiple data phases. At a PCI clock frequency of 33 MHz, 132 Mbytes/sec of transfer rate may be achieved with 32-bit implementation.

An extensive and detailed version of the material presented here can be found in [Ref. 10].

There are two participants in every PCI transfer: the initiator and the target. The initiator is the device that initiates the transfer; it is also generally serves as the bus master. The target is the device currently addressed by the initiator for data transfer.

A PCI physical device package may be in the form of a component integrated onto the system board or may be implemented on a PCI add-in card. Examples of a typical device would be a network, display or SCSI adapter. Each PCI package may incorporate from one to eight separate functions. A function is represented as a logical device. A package containing one function is referred to as a single-function PCI device while a package containing two or more PCI functions is referred to as a multi-function PCI device. Each function contains its own, individually addressable configuration space, which

is 64 double words in size. Its configuration registers are implemented in this space. Using these configuration registers, configuration software can automatically detect the function's presence; determine resource requirements and can assign resources to the function. For example, Windows NT operating system (O/S) maps devices to memory addresses during its initialization. Therefore, the O/S knows how and where it can communicate with the devices. These assignments are absolutely dynamic, a network adapter with assigned resources might not be assigned to the same resources at the next initialization.

All actions on the PCI bus are synchronized against the PCI clock signal (CLK). The frequency of the CLK signal may be anywhere from 0 MHz to 33MHz.

Every PCI transaction starts with an *address phase* that is one PCI clock period in duration. The initiator identifies the target device by driving the start address onto the PCI address/data bus and the transaction type by driving the command type onto 4 bit wide PCI Command/Byte enable bus. The initiator also asserts a frame signal (FRAME#) to indicate the legitimacy of the data on buses. When a PCI target determines that it is being addressed, it must claim the transaction by asserting the device select (DEVSEL#) signal. By decoding the address latched from the data bus and the command type from the command bus, a target device can determine if it is being addressed and the type of transaction requested.

The *data phase* of a transaction is the period during which a data object is transferred between the initiator and the target. The number of the bytes to be transferred is determined by the number of Command/Byte Enable signals that are asserted by the initiator during this phase. Each data phase is at least one PCI clock period in duration. Both the initiator and the target must indicate that they are ready to complete a data phase;

otherwise, the data phase is extended by a wait state one PCI clock period in duration. The PCI bus defines ready signal lines used by both the initiator (IRDY#) and the target (TRDY#) for this purpose. The initiator does not issue a transfer count to the target. Rather, in each data phase it indicates whether it's ready to transfer the current data item and, if it is, whether it is the final data item. FRAME# is asserted at the start of the address phase and remains asserted until the initiator is ready (asserts IRDY#) to complete the final data phase. When the target samples IRDY# asserted and FRAME# deasserted in a data phase, it realizes that this is the final step of the data phase. However, the data phase will not complete until the target has also asserted the TRDY# signal. When the last transfer has been completed, the initiator returns the PCI bus to the idle state by deasserting its ready line (IRDY#). Hardware keeps the data transfer within bounds.

At a given time, one or more PCI bus master devices may require use of the bus to perform data transfer with another PCI device. To allocate the ownership of the bus to a particular master there is a need for a regulation mechanism. This mechanism is called the *arbitration mechanism.* Each master device is connected to the arbiter via separate pair of request (REQ#) and grant (GNT#) signals. So, the device that needs the bus will ask for the use of bus from the arbiter. The PCI specification does not define the scheme used by arbiter, so it is implementation-dependent (generally the arbiter is implemented in the host-PCI bridge on the motherboard), but the specification states that it should be a fair algorithm. Another issue in bus arbitration is the maximum latency, meaning that when the requesting master takes the ownership of the bus, it cannot keep the bus forever; there is a maximum time it can keep the bus. The startup configuration software can determine the priority to be assigned to each member PCI device by reading from Maximum Latency

configuration register associated with each bus master. The bus master designer hardwires (set it to a value that cannot be changed) Maximum Latency register to indicate, in increments of 250 ns, how quickly the master requires access to the bus. Each requesting master asserts its REQ# output. As a response to assertion of REQ#, if the arbiter grants the request, then it asserts the master's GNT# pin. This set-up is actually for just one transaction. If the master has another transaction to perform immediately after the one it just initiated, it has to keep the REQ# line asserted still after it asserts FRAME# to begin the transaction. The master keeps its FRAME# asserted while the current transaction is still in progress. It deasserts it when it is ready to complete the final data phase. However, this does not guarantee that the next request will be granted. Depending on the pending requests, the arbiter may or may not permit the master to maintain bus ownership. When the ownership is not maintained, the master should keep the REQ# line asserted until it successfully acquires the bus again. The arbiter may deassert a master's GNT# on any PCI clock. When this occurs, the transaction must not proceed.

The PCI bus arbitration scheme allows the bus arbitration to take place while the current initiator is performing a data transfer. If the arbiter decides to grant ownership for the next transaction to another master, it removes the GNT# from the current master and issues it to the next master. However, the next master cannot assume bus ownership until the bus is idled by the current master. This is called hidden arbitration.

Up to this point, regular PCI read/write cycles have been explained. There are also other PCI cycles for configuration, broadcasting, interrupt acknowledging purposes:

Interrupt Acknowledge, is used to handle interrupt acknowledgements by the processor. Therefore, if a PCI device generates an interrupt, it can be acknowledged that the host processor knows about the interrupt and it is processing it.

The Special Cycle command is issued when an initiator wants to broadcast a message to one or more targets residing on a target PCI bus.

I/O Read and Write is used to transfer data between the initiator and the currently addressed I/O target. The PCI specifications define five commands to access memory: Memory Read, Memory Write, Memory Read Multiple (optional), Memory Read Line (optional), Memory Write-and-Invalidate (optional). The target device may support optional commands.

Configuration Read and Write allow each PCI function to implement up to 64 double words of configuration registers that are used during system initialization to configure the logical device for operation. To access these registers configuration read and write commands are used. To access a device's configuration space, a configuration read or write must be initiated and the addressed device must sense its ID select (IDSEL), a signal similar to chip select, input asserted during the address phase. Three bits in Address/Data bus (AD[10:8]-bits 8,9,10), select the function within the package and six bits in the Address/Data bus (AD[7:2]-bits 2,3,4,5,6,7) are used to select the registers of the target function's configuration space.

(1) PCI Issues Related To The TCBE. The TCBE will be an intermediate card between the HAS and the client PC. It will be an add-on card and will be designed to be plugged into the PC's PCI extension ports. There is already a well-

established interface for PCI devices. We will design the TCBE with strict adherence to PCI specifications.

The TCBE must have PCI master device capabilities since it will incorporate devices of its own on its own PCI bus. Therefore, the TCBE will provide PCI configuration address space to allow the host system to configure the TCBE and incorporated devices. Basically, the TCBE will provide configuration registers required for a PCI master according to the PCI specifications.

3. TCBE Interface Presented to the High Assurance Server

The following section describes network communication and network access protocols, to give reader a brief understanding of protocols especially TCP/IP and CSMA/CD since both protocols will be used to access the network and communicate within the MLS LAN. The section concludes with protocol issues related to the TCBE.

a. Network Protocols – TCP/IP

Detailed explanations of network protocols can be found on numerous resources[Ref. 15][Ref. 16]. As today's most popular network protocol, TCP/IP is a major subject in many of those resources, it is also possible to find many TCP/IP dedicated resources. The information below is given to brief the reader about the TCP/IP and can be found in [Ref. 12].

Two protocol architectures have served as basis for the development of interoperable computer communication standards, The transmission control protocol/internet protocol (TCP/IP) protocol suite and the open systems interconnect (OSI) reference model. Although the OSI model has become the standard model for classifying

communication functions, TCP/IP is the most widely accepted and is the most widely used architecture in real systems.

The OSI model was developed by the International Organization for Standardization (ISO) as a model for computer communication architecture and as a framework for developing protocol standards. It is a clear design with layered functions. It has seven layers:

- The Application layer provides access to the OSI environment for users and provides distributed information services.
- The Presentation layer provides independence to applications from differences in data presentation such as endian differences.
- The Session layer provides the control structure for communication between applications. It establishes, manages, and terminates sessions.
- The Transport layer guarantees reliable, transparent transfer of data between end stations, provides end-to-end error recovery and flow control.
- The Network layer provides upper layers with independence from the data transmission and switching technologies used to connect systems. It is responsible for establishing, maintaining and terminating connections.
- The Data Link layer provides for reliable transfer of information across the physical link. It sends data as blocks (frames) with necessary synchronization, error control and flow control information.

- The Physical layer is concerned with transmission of bit stream over physical medium. It deals with the mechanical, electrical, functional, and procedural characteristics to access the medium (cable).

TCP/IP is a result of protocol research and development conducted by the experimental packet switching network, ARPANET, funded by Defense Advanced Research Projects Agency (DARPA). It is generally referred to as the TCP/IP protocol suite since it is consists of a large number of protocols. There is no official TCP/IP protocol model as there is the case of the OSI model. However, based on protocol standards that have been developed, one can organize the TCP/IP as five relatively independent layers such as OSI's:

- The Application layer provides communication between processes or applications on separate hosts.
- The Transport layer provides end-to-end data transfer. It hides the details of underlying networks from the application layer.
- The Internet layer is concerned with routing the data from the source to the destination through one or more networks connected.
- The Network Access layer defines characters of the transmission medium, signaling rate and signal encoding scheme.
- The Physical layer handles the physical interface between a data transmission device and a transmission medium or network.

Figure 15 illustrates the layers of both OSI and TCP/IP and showing roughly the correspondence in functionality between the two.

For networked communications, some sort of network access protocol, such as Ethernet, is required. TCP/IP enables the host to send data across the network to another host or, in case of another host on another network, to a router. IP is implemented in all of the end systems and the intermediate nodes such as routers that are acting as relays to move blocks of data from one network to another. TCP, on the other hand is implemented only in the end systems: it keeps track of the blocks of data to ensure they all are delivered reliably to the appropriate application.

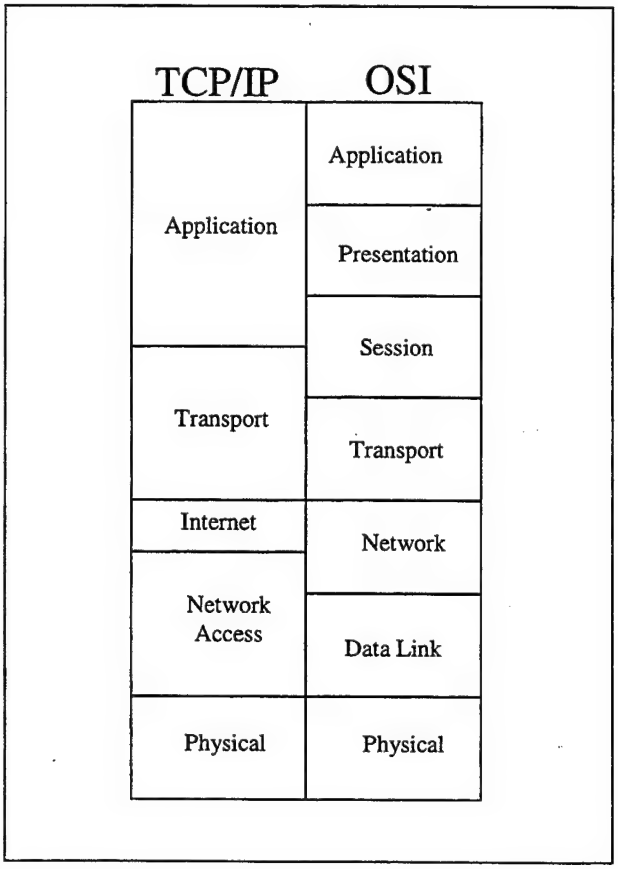

Figure 15. Network Protocols [Ref. 12]

To be able to communicate, every entity in the system must have an unique address. For TCP/IP, two levels of addressing are needed. Each host on a network must

have a unique global inter-network address; its so called IP address, which allows the data to be delivered to the proper host. Each application or process running on a particular host must also have an identifier that is also unique within that host which allows host-to-host protocols (TCP) to deliver the data to the target application. This latter address is called the port.

Here is an example of an operation. Suppose that a process, associated with port 1 at host X, wishes to send a message to another process at port 2 at host Y.

The process at X hands the message down to TCP with instructions to send it to host Y, port 2. To easily control and manage the operation, TCP breaks data into blocks and appends control information, known as the TCP header, which generally includes the destination port, the sequence number of the packet, checksum information etc. Then TCP hands the message packets to IP with instructions to send it to host Y (IP does not need to know port number). The data segments must be transmitted across one or more networks and relayed through one or more intermediate nodes. This operation requires the use of control information. Thus, IP appends an IP header to each segment to form so called IP datagram. Next, IP hands the message composed of packets down to network access layer such as Ethernet with the instruction to send it to the first hop on the way to host Y. This may be a network gateway or a router. Finally, the network access layer appends its own header and creates a packet or a frame. The packet is sent across the network to the first hop. The packet header contains the information that the network needs to transfer the data across the network to the destination network address. At any intermediate node such as router W, the packet header is stripped off and IP header examined. According to the

destination address in the IP header, the node redirects the data across the network together with a new network access header.

When the station Y receives the data packets, the reverse process starts. At each layer, the corresponding header is removed and the remainder is passed on to the next higher layer until the original data are delivered and reconstructed at the destination process. Examples of application layer protocols are Simple Mail Transfer Protocol (SMTP-port 25), File Transfer Protocol (FTP-port 20/21), Telnet (port 23), and Hyper Text Transfer Protocol (HTTP-port 80).

(1) TCP/IP Issues Related To The TCBE. Since the TCBE will reside in between the HAS and the PC, there will be an interface to the high assurance server. In the MLS LAN the network communication protocol will be standard TCP/IP. However, to provide additional control over communications, standard port numbers will be remapped by the TCBE.

The TCP/IP interface to the HAS will be through a standard NIC on the TCBE. During the SAK state, the TCBE will directly communicate to the TCB on the server. For regular communications, the TCBE will supervise the communications and will either block or allow the data flow according to its current system state. Additionally the TCBE will remap the standard TCP/IP port numbers to the non-standard ports since the system designed by Breyer-Joyner and Heller (IMAP server ported to the XTS-300) [Ref. 12] uses these non-standard ports to communicate to the high assurance server.

a. Network Access Protocols-CSMA/CD

Network adapters enable communication between differing systems across a network. Depending on the distance between the stations networks are classified as Local Area Networks (LANs) or Wide Area Networks (WANs).

Over time, three different network topologies, which are also methods for connecting the stations together, for LANs have been developed; Bus, Star and Ring. In all topologies, the data are transferred in the form of varying sized packets. In addition to actual data, packets also contain frame data for identifying the beginning and the end of the packet, error checking code and also the address of the target as a minimum.

Naturally, the individual stations of a network cannot access the network without restrictions and regulations. Therefore, a set of rules is essential. There are two prevalent access procedures in association with LANs.

Carrier Sense Multiple Access/Collision Detection (CSMA/CD) is the most popular access protocol used in today's local area networks. Here how it works:

As only one line connects the computers, only one station can send data at a time. However, before station X can transmit anything, it must find out whether or not another station Y is currently occupying the line. For this reason station X senses whether a carrier signal is present on the line. If so, then the station X must wait and retry later, if it is not occupied, station X can start transmitting the data. In some cases, it is possible that another station, W, is also preparing to transmit data. And it is possible that at the time it is checking the bus, there is no carrier signal and the bus is free. Therefore, stations X an W will simultaneously try to transmit data and will cause a multiple access and the data packets will disrupt or cancel each other. So, a collision must be recognized. To detect

collisions, the transmitting station continues to monitor the network after transmitting the data, thus it receives its own feedback from the network to detect a collision. If a collision is detected, a signal is broadcast by the detecting station to inform the other stations that there has been a collision. All the stations immediately stop sending data, so the bus is cleared and becomes free again. A counter is started with a randomly selected initial value on every station that wishes to transmit data. When the counter of a station reaches the end of counter, the station restarts attempting to acquire the bus to transmit.

Token passing is another method of communicating between the computers. It utilizes a data structure for controlling network access, called a token. The token runs continuously around the ring of computers, passed from station to station. The token actually represents a permit to access the network, only the station holding the token may address the ring and therefore transmit any data. Station X holds and passes the token to the next station with data. The next station passes the token until the target station is reached. The target station receives the data, empties the token and passes the empty token to the next station. This goes on until the token returns to the sender station to acknowledge that the packet has been received.

Owing to Ethernet, the CSMA/CD access procedure has become widely used. Xerox developed it in 1976. It generally uses the bus topology. The maximum transfer rate of Ethernet is 100 Mbits/sec.

(1) Ethernet Issues Related to the TCBE. The access to the network will be accomplished through an Ethernet NIC. There is no emulation needed for the TCBE to interface with the network since the network will be an Ethernet network. The NIC will directly interface to the network physical media by using Ethernet access protocol.

The TCBE will provide regular network access through the NIC and it can perform additional functions such as encryption and integrity checks (checksums).

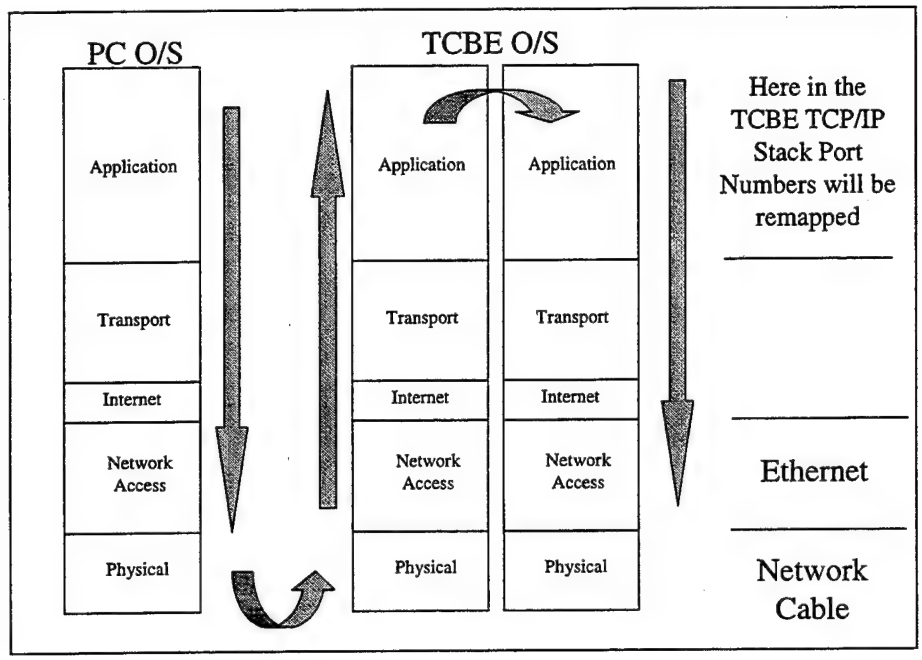

Figure 16. The TCBE Network Operation

In Figure 16, the relation between the TCP/IP, the Ethernet and the TCBE is shown. The network packets on the client PC are processed as if they will destined to a standard CSMA/CD network interface card. The packets will be sent to the TCBE instead of NIC since the TCBE emulates the NIC on the client PC. Then the TCBE will process these packets if they are received from network. The packets headers will be stripped off, when the packets arrives at the application layer, the TCBE O/S will start to prepare the packets for network again with the new port numbers. The Ethernet protocol will reside within the TCP/IP protocol as the network access protocol. The TCP/IP network packets will be appended Ethernet headers to be sent their target addresses within the network. If the

packets are destined to another network then the IP protocol will allow the packets to find their destination addresses.

C. INTERNAL INTERFACES

1. TCBE Interface Presented to the Hard Disk Controller

The following section briefs the reader about the hard disk controller technologies being used today's computers. The section concludes with a description of possible TCBE hard disk controller.

a. Hard Disk Technologies

Hard disks are the secondary main repository for computer systems. Starting with the very first PC, there have been different hard disk technologies together with their interface architectures to PC's motherboard. Although there have been many hard disk architectures, there are two different main architectures for hard drives: Intelligent Drive Electronics (IDE) and Small Computer System Interface (SCSI). The rest of the architectures are enhanced versions of these two standard architectures.

IDE is the most used hard disk interface standard for PCs. The drive itself has only electronic elements required to drive the motors and gates of the physical drive. A separate adapter called hard disk controller, which is remote from the drive and connected to the drive via a cable, carries out the control of executing commands. Thus, the drive itself is non-intelligent. Because of the long signal path, high data transfer rates fail and IDE is considered a slow standard. To overcome the problem, new IDE hard disk drives integrate the controller into the driver to eliminate the long signal path and propagation delays.

IDE is a logical interface between the PC system and the hard disk drive. The physical connection between the bus and the IDE interface of the drive(s) is established by a host-adapter. The host adapter accommodates a few buffers, decoder logic and the controller. An IDE interface can serve a maximum number of two drives, one of which is the master and the other is the slave.

The main advantage of IDE is the very simple structure of the host adapter. For this reason the IDE drives generally have a price advantage over other technologies.

A very flexible and powerful option for connecting hard drives to a PC is the Small Computer System Interface (SCSI) standard. SCSI defines a bus between a maximum of eight SCSI units. SCSI drives are intelligent and the disk controller is always integrated on to the drive. Therefore, they are more complex than IDE drives. To connect a SCSI drive to a PC, a SCSI host adapter is required. The SCSI host adapter is itself a SCSI device and is quite complex. Because the adapter is a SCSI device, seven more SCSI devices can be connected to a PC.

Very high data rates can be achieved with SCSI drives. SCSI bus serves for a data exchange among the SCSI units connected. A maximum of two units may be active at any time in a SCSI bus. The data exchange can be carried out between the host adapter and a drive or between two drives. The data exchange can be accomplished without CPU intervention. Because of the complexity of the host adapter and SCSI bus, SCSI drives are more expensive than IDE drives and yet they are the choice of high speed and more reliability than IDE with higher cost.

(1) The TCBE hard disk controller. As explained in Chapter II, the hard disk controller will be incorporated on to the TCBE. The TCBE will need to

present a regular hard disk to the host PC operating system. It can be accomplished in two ways:

One way is to initialize the TCBE as a PCI-to-PCI bridge and let the host operating system probe and initialize the hard disk controller and hard drives on the TCBE's PCI bus and later gain the control of the controller by modifying the PCI configuration registers within the TCBE, therefore, grant/deny the host operating system's access to the NIC. This way does not require any custom device driver for the TCBE hard disks.

The other way is to hide the controller and the drives from the PCI configuration cycles and initialize them. In this case, the TCBE will need a NIC driver of its own that will run within the TCBE operating system. Then the TCBE will emulate the NIC on the host operating system side. An intermediate driver, redirector, should be inserted between the TCBE and the PC O/S. Therefore, a TCBE hard disk driver should be written and incorporated into the TCBE operating system (see Figure 17).

According to the requirements, the TCBE will ghost the operating system image from a read only source to one of the read/write hard disks for each new session (see detailed discussion of delivery methods of operating systems in [Ref. 6]). After accomplishing this, the TCBE will provide a regular system hard disk interface to the host PC operating system. The TCBE hard disk interface will be a standard PCI SCSI hard disk controller to be easily integrated into the TCBE's PCI bus.

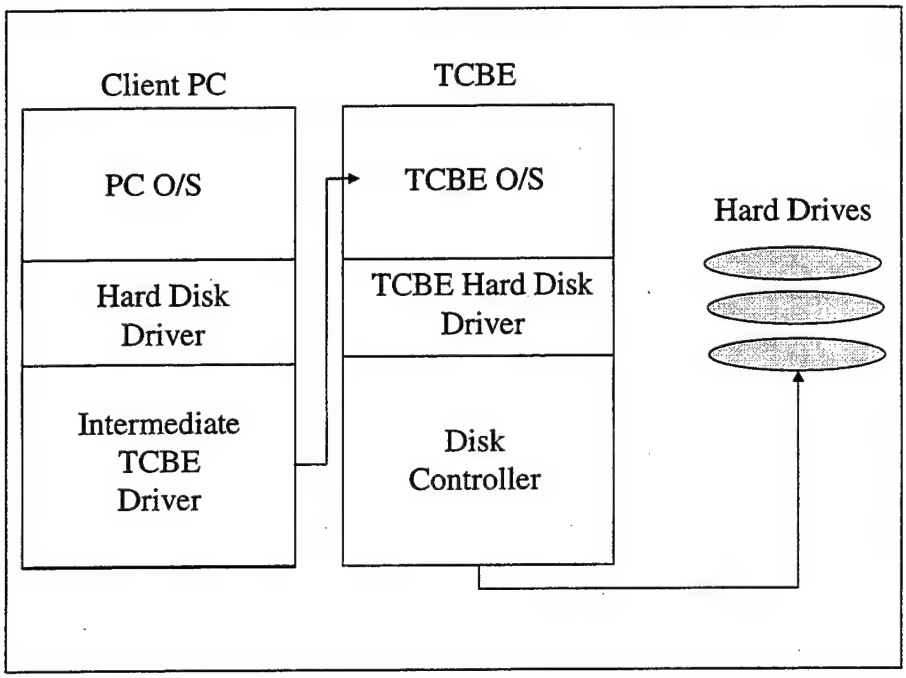

Figure 17. The TCBE Hard Disk

2. TCBE Interface Presented To The Network Interface Card

We will present the interaction between a typical operating system and the network interface card in the following section. The function of a network adapter driver and the operating principles of a standard network interface card are described. The material presented is an example of a standard network adapter and can be found on 3COM 3C90x and 3C90xB NICs Technical Reference [Ref. 13]. The section concludes with a description of TCBE NIC interface.

a. How does the network interface card work?

Access to the network is accomplished via components called Network Interface Cards (NICs). In a PC system, the operating system controls the NIC and accesses

the NIC by using software called a driver. Here is a brief presentation of how NIC drivers and NICs operate:

To move data between the PC and the NIC, drivers setup data structures in the system main memory to specify buffers to be used for packet data movement. These structures, called descriptors (Upload/download packet descriptors), are linked together in the memory to form lists. All packet data is moved to/from the NIC by bus operations. In the MLS LAN project the NIC will be PCI-compliant, so all data movements will be handled by PCI bus master operations. The NIC also uses bus master operations to read descriptor information out of system main memory and to write information's status back into descriptors.

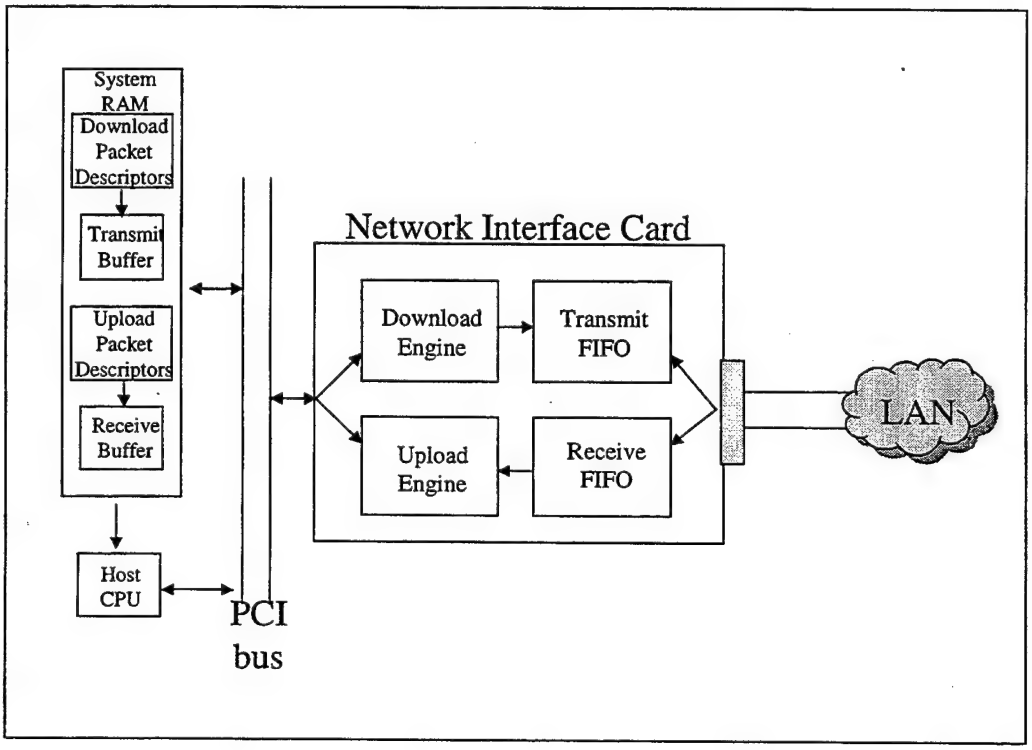

Figure 18. A Network Adapter Block Diagram [Ref. 13]

Figure 18 shows a block diagram. Movement of a transmit packet to the NIC is called download. The list of download packet descriptors (DPDs) is called the downlist. Similarly, received packet movement is called an upload and the list of upload packet descriptors (UPDs) is called uplist. The device driver creates and maintains the uplist and the downlist. It starts the download process by writing the address of the first download descriptor in the downlist to download pointer register. Uploads are started by writing the first upload descriptor address to the upload pointer register.

The packet to be transmitted is first placed in data fragments in the system memory. Next, a list of DPDs that point to the fragments is also created in memory. The head of the list is the DPD that corresponds to the current download packet. The download list pointer register points to this DPD. As the DPD is processed, the fragment address and fragment length values are fetched one by one from the DPD into on-NIC registers, which are used to control the data download operations.

The NIC exits from the reset state with the download engine in the idle state, ready to start processing a downlist as soon as a nonzero value is written into download pointer register. The NIC fetches the information and moves the packet data into transmit FIFO. The NIC initiates packet transmission as soon as either the entire packet is resident in the FIFO or the number of bytes that are resident is greater than a configurable value.

The packet upload mechanism is similar to the download mechanism. Upload is structured around a linked list of packet descriptors, called upload packet descriptors (UPDs). UPDs contain pointers to the fragment buffers into which the NIC is to place receive data. The linked list of UPDs is called the uplist. The head of the uplist is the UPD that corresponds to the current upload packet. The uplist pointer register points to this

UPD. As the UPS is processed, the fragment address and fragment length values are fetched one by one from the UPD into on-NIC registers (FIFOs), which are used to control the data upload operations. When the NIC exits from the reset state, the upload engine is in the idle state and ready to start processing an uplist as soon as a nonzero value is written into upload pointer register.

(1) The TCBE Network Interface Card. As in the case of hard disk interface, the TCBE will need to present a regular network interface to the host PC operating system. At the same time, the NIC has to be totally controlled by the TCBE. There are two ways to accomplish this goal;

One way is to initialize the TCBE as a PCI-to-PCI bridge and let the host operating system probe and initialize the NIC on the TCBE's PCI bus and later regain the control of the NIC by modifying the PCI configuration registers within the TCBE, therefore, grant/deny the host operating system's access to the NIC.

The other way is to hide the NIC from the PCI configuration cycles and initialize the NIC. Therefore, the TCBE will need a NIC driver of its own that will run within the TCBE operating system. Then the TCBE will emulate the NIC on the host operating system side (see Figure 19).

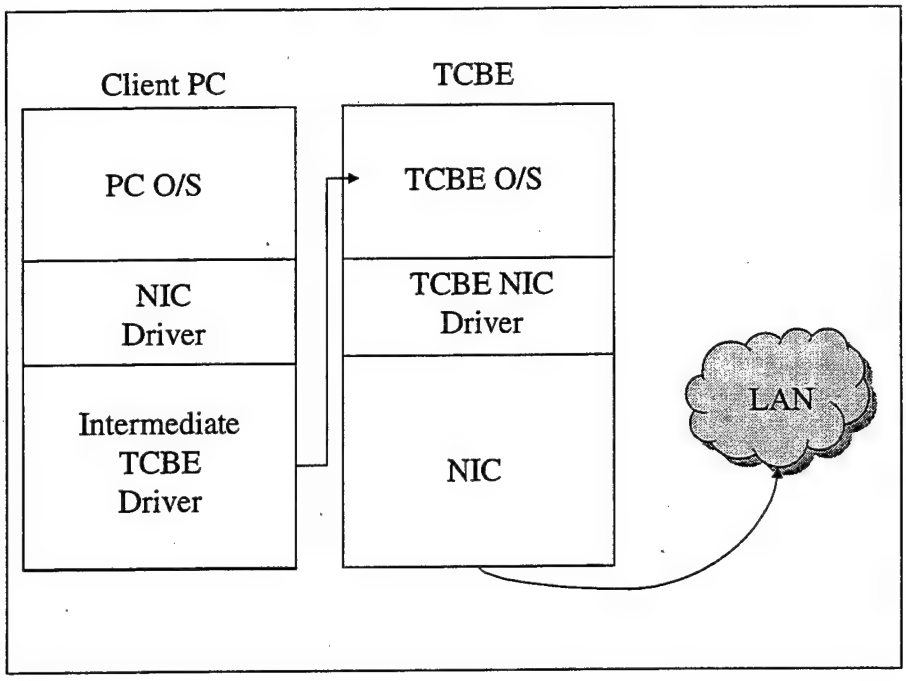

Figure 19. The TCBE Network Interface Card

D. CONCLUSION

In this chapter, we analyzed the TCBE interfaces. The analysis presented an overview of the underlying technologies and the implementations of selections of those technologies for the TCBE. The analysis in this chapter will lead us to the TCBE prototype. In the next chapter, we will present a preliminary TCBE prototype and we will present a security analysis for the prototype together with some experiments designed to validate the security analysis of the TCBE prototype.

THIS PAGE INTENTIONALLY LEFT BLANK.

IV. PRELIMINARY ANALYSIS OF TCBE DESIGN AND CONSTRUCTION

This chapter will discuss design issues related to construction of the TCBE. There are two different approaches to the design of the TCBE. A TCBE can be designed from the scratch or it can be constructed by putting the right pieces together, as an integration effort. Although designing from scratch may result in a compact, fast, effective TCBE, using commercial pieces and integrating them will give us the advantage of focusing on the TCBE's functionality instead of hardware specific issues such as PCI compliancy. This permits rapid prototyping of the TCBE. Thus, our choice is to construct the TCBE by integrating readily available components. The key to successful integration is to find the right components, integrate them and make them work together.

We have already defined the major components of the TCBE. The TCBE will have a processor, RAM/ROM (including Flash) memory, a keyboard, a display, a hard disk controller and accompanying hard disk(s), a network interface card and related circuitry for the above components.

A. PROTOTYPE HARDWARE

We will construct the TCBE prototype by trying to integrate readily available, appropriate commercial components.

To avoid implementing the TCBE from scratch and to try to comply with PCI specifications, we need the TCBE to be a PCI device with master capabilities. We will extend the PCI bus and provide the TCBE with its own PCI bus. The TCBE will be the

owner and the master of that bus. The TCBE needs to have a processor, memory and a standard PCI bus interface so that standard components such as a standard PCI network interface card or a standard PCI SCSI disk interface can be easily integrated. We will try to integrate as many components as an add-on PCI card . For the components that are not available to be integrated as an add-on card, we will modify the TCBE board. Nevertheless, the effort will focus on integration of the components and making them work together as required. For fast, easy experimental development, we will use a developer's kit.

The initial TCBE prototype will consist of the following components:

- An Intel i960 RM I/O processor board,
- A Standard PCI 3COM 10/100 Mbits XL Ethernet card.

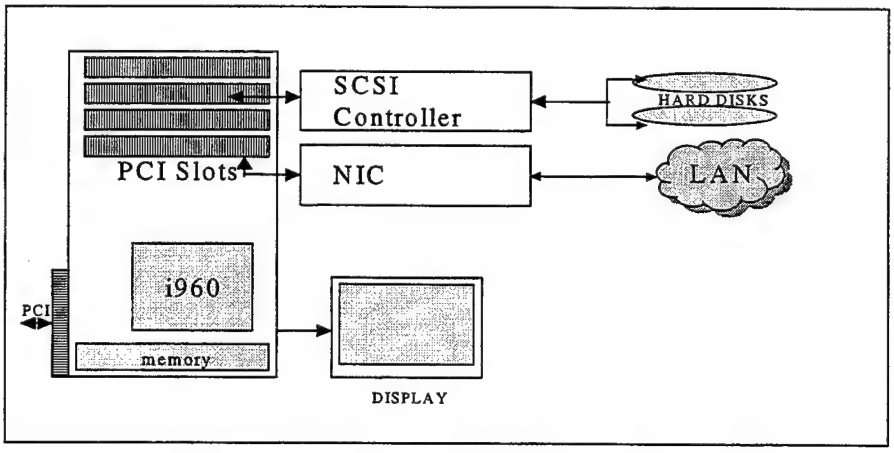

Figure 20. The TCBE Prototype

The Intel i960 board is an intelligent I/O board with PCI master capabilities. For the prototype, we will use i960 RM developer's kit. The board contains an Intel 80960JT reduced instruction set computer (RISC) processor, a PCI-to-PCI bridge, volatile/non-

volatile memory, and four PCI expansion slots. We will plug a standard 3COM PCI network adapter and a standard PCI SCSI controller directly to the i960 board.

In the following sections, the details about the Intel board will be given and the board will be analyzed and tested for the feasibility of hardware support in terms of non-by-passability and tamper resistance to build a TCBE.

B. INTEL I960 RM I/O BOARD

The Intel i960 RM board is a multi function PCI compliant device designed for developing intelligent I/O peripherals such as ATM cards, satellite communications cards etc. It is a PCI-to-PCI bridge, an address translation unit and a RISC processor. The PCI-to-PCI bridge is a path between two independent PCI buses and can extend PCI architecture and provide ability to overcome PCI electrical load limits. In addition to bridge functions, the i960 RM board includes a 80960 JT core processor to bring intelligence to the bridge, such as forwarding of PCI transactions to the i960 memory instead of to the secondary PCI bus. The 80960 JT processor is an Intel RISC processor being heavily used for embedded applications with one instruction per clock execution rate. A local bus integrates processor, local memory and I/O. (Figure 21)

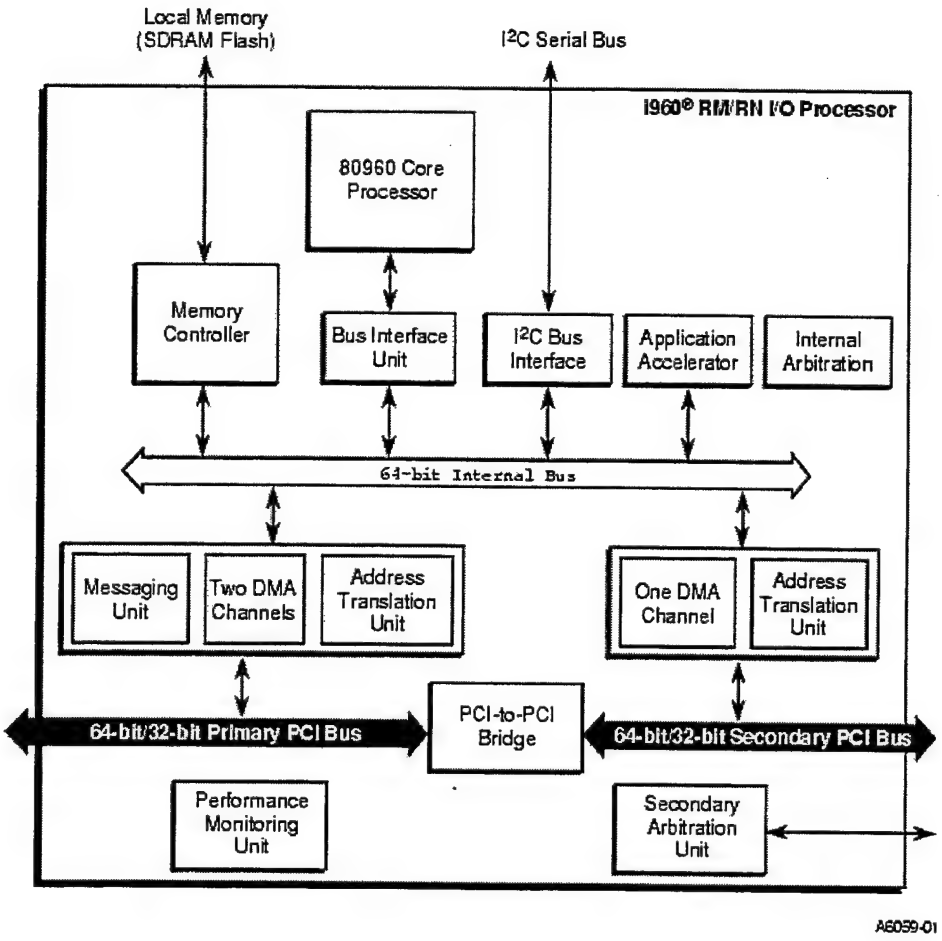

Figure 21. i960 I/O Processor Block Diagram [Ref. 17]

As shown in Figure 21, the PCI-to-PCI bridge connects the Primary PCI bus to the Secondary PCI bus directly. But primary and secondary address translations units can also be used to connect two PCI buses indirectly via internal bus which allows the 80960 core processor to process the data in between two buses.

As shown above in the block diagram, the i960 RM I/O board has a DMA controller, address translation units, messaging unit, arbitration unit, application acceleration unit, bus interface unit besides its core processor and accompanying memory.

The i960 board provides two ways of communication between the two PCI buses on each side of the board. The default mode is to function as a PCI-to-PCI bridge, which actually means to transfer data from one bus to another directly in either direction (Figure 22).

Figure 22. PCI-to-PCI Bridge (direct transfer) [Ref. 17]

The second way requires the involvement of the bridge and the address translation units. Data from either bus are again transferred from one bus to the other but they are transferred indirectly, going through the processor's internal bus. This allows the i960 processor to process the data in between the buses (Figure 23)

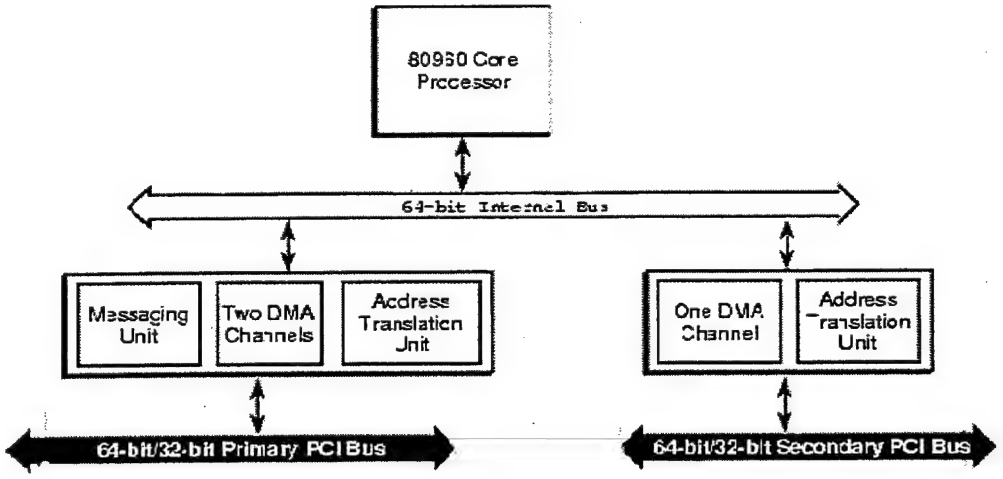

Figure 23. Address Translation (Indirect transfer) [Ref. 17]

The bridge and address translation unit can also work together and can hide PCI devices on the second bus from PCI configuration cycles and allow these hidden devices to

use a private address space. The i960 actually blocks the devices on the secondary bus from the PC O/S PCI configuration cycles.

The DMA controller provides high-speed data transfers between PCI devices and on board local memory. There are three separate DMA channels: two for primary PCI bus, one for the secondary PCI bus.

The address translation unit allows PCI transactions direct access to local memory. The i960 RM processor has direct access to both PCI buses. The address translation unit provides transactions between the PCI address space and the 80960 RM processor address space. Address translation unit is controlled through programmable registers.

A messaging unit provides data transfers between the PCI system and the 80960 RM processor. The messaging unit uses interrupts to notify the PCI system or the 80960 RM processor when new data arrives. It has different messaging mechanisms that are configurable such as message registers, a doorbell mechanism, and circular queues. Each mechanism allows a host processor or external PCI device and 80960 processor to communicate through message passing and interrupt generation.

The memory controller unit allows direct control of local memory with address translation unit configuration registers. Local memory can be configured as PCI addressable memory or private processor memory.

The arbitration unit provides PCI arbitration for the secondary PCI bus. The arbitration includes a fairness algorithm with programmable priorities and six external PCI request and grant signal pairs.

The bus interface unit provides an interface between the 100 MHz 80960 JT processor and the 66 MHz internal buses.

The 80960 JT core processor is a 100 MHz RISC processor that sustains an instruction per clock. It has an independent multiply/divide unit. It is a superscalar processor with a multiple stage pipeline. It has separate instruction and data caches. It supports bit, integer, ordinal, triple and quad word data types and provides absolute, register indirect, indexed and indexed with displacement addressing modes.

The i960 RM processor has two independent 32-bit timers capable of counting at several clock rates and generating interrupts. It has a priority interrupt controller, which allows prioritization of interrupts from interrupt level 0 to 31.

C. TCBE OPERATION AND SECURITY ANALYSIS OF I960 HARDWARE

In this section we will present the operational modes of he TCBE and according to these modes, we will analyze the i960 board. We will examine whether the i960 board can provide the required hardware support for the TCBE to be non-by-passable and tamper resistant.

The TCBE will operate in two different operation modes;

- The TCBE as a bridge and,
- The TCBE as a PCI custom device (gatekeeper).

When the secure attention key (SAK) is pressed, the TCBE should switch from bridge mode to gatekeeper mode.

The i960 board is a PCI device; moreover, one of the board's peripherals is a PCI-to-PCI bridge. Therefore, it has to provide a set of PCI configuration registers to comply with the PCI specifications. These registers lie in a memory space called the PCI configuration space, which is accessible, by the i960 processor as well as by the host PC processor. For the PCI-to-PCI bridge to operate and for the host OS to detect, configure and initialize devices beyond the bridge, the host PC probes into this configuration space, and allocates resources (IRQ, memory) by programming these registers during the initialization process. This is the mode that the i960 operates and allows the host processor to configure i960 peripherals. This mode is called mode 0 [Ref. 14: p.11-3].

After detecting and configuring the bridge and the devices, the host PC OS loads the drivers for the bridge and the devices on the i960 board as well. Therefore, the devices are all registered and operational at that point. The OS can reach, and communicate with the devices, and can perform PCI reads and writes.

Operating in mode 0, the i960 board behaves like nothing more than a standard PCI-to-PCI bridge. The client PC is free to operate as a regular desktop computer and requires no association with the HAS. For example, the user can surf the web, send e-mail but not access the HAS.

However, when the SAK is pressed, the TCBE can take control of the data flow to and from the network by shutting down the PCI-to-PCI bridge. This can be accomplished by defining the memory base register and the memory limit register to have the same value, therefore, defining a closed bridge [Ref. 14: p.14-17]. The registers, which define the address space where the bridge exists and the registers that control the operation of the

bridge are memory mapped, programmable, and accessible by both the host PC processor and the i960 processor [Ref. 14: p.14-16][Ref. 14: p.3-8]. However, the host PC processor can only probe and modify these registers by performing a Type 0 PCI configuration cycles [Ref. 14: p.14-6]. PC systems generally perform PCI configuration cycles to setup the devices during system initialization and/or during OS initialization.

On the other hand, the i960 processor can program these registers anytime. Therefore, when the SAK is pressed, the i960 can shutdown the bridge anytime. It is technically possible for the host system OS or a program to reopen the closed-bridge by modifying the bridge control registers back to their original values. Although, OSs do not perform PCI configuration cycles other than at system initialization, a clever adversary using a malicious code can perform a PCI configuration cycle and regain the control of the bridge.

This regain of control must be thwarted and the PCI specifications define a register that does not allow the host PC processor to configure a PCI device. The extended bridge command register (EBCR) has a "configuration cycle retry" bit, which determines whether the PCI device is to accept or to reject the configuration commands from the primary interface [Ref. 14: p.14-7].

Therefore, when the SAK is pressed, the TCBE can shutdown the bridge and set the configuration cycle retry bit in EBCR to prevent host PC OS or any other malicious software from modifying the PCI configuration registers and gaining control of the bridge. This proves that the TCBE is non-by-passable.

After the control of the data flow is established, the TCBE can perform its tasks to establish a communications channel between the user and the trusted computing base (TCB). Upon completion of a successful login to the TCB, the client PC will be purged (restarted) and the TCBE should switch to the TCBE as a gatekeeper mode.

The i960 board can operate in another mode called mode 3. In mode 3, the i960 processor initializes and controls the i960's initialization before the host processor is allowed to configure i960 peripherals [Ref. 14: p.11-3].

In this mode, the TCBE will not allow the host OS to probe or initialize the bridge or the devices beyond the bridge. Therefore, the host OS will not be aware of the devices on the TCBE PCI bus. In other words, the i960 will hide these devices [Ref. 14: p.1-2]. The TCBE will have to initialize these devices, which means the TCBE OS will need to run the device drivers for these devices. At the same time, the TCBE will need to emulate these devices and present theses devices to the host OS as if they actually exist on the TCBE. This means that a custom TCBE driver is required that will handle and redirect all network-destined communications to the TCBE. In this mode, the client OS is not aware of the devices on the TCBE PCI bus, since the NIC is on the TCBE bus, there is no channel between the client PC and the network. This is shows that the TCBE is the only channel to the network and it is non-by-passable.

We leverage the i960's ability to hide devices from PCI configuration cycles and to allow these devices to use a private PCI address space. The address translation unit can generate internal PCI configuration cycles to configure these devices [Ref. 14: p.1-2]. Additionally, by emulating devices the TCBE will receive the PCI transactions destined for

these devices and will translate these PCI address space transactions to i960 internal address space transactions where they can be processed by device drivers running on the TCBE OS. Having the ability to buffer the transaction data in the i960 internal memory also allows the TCBE to perform additional functions such as encrypting, decrypting, checksuming the data.

The i960 memory can be mapped to read-write, read-only or memory mapped I/O. The i960's memory can be divided into pages or can be access restricted (supervisor or user) to certain areas in the memory to protect the TCBE kernel [Ref. 14: p.3-10]. Moreover, since the TCBE OS will be served from i960 ROM, it can be concluded that the TCBE OS will boot to a known state and we will have the assurance that it is not modified. However, after the OS image is loaded into the RAM memory, it is possible to modify the OS. Nevertheless, the ability to configure i960 memory as a read-only will be solve this problem by configuring the portions of memory where the TCBE OS will be kept as read-only and guarantee that the OS can't be modified. Therefore, the TCBE OS, the mechanism that will support the high assurance server's security policy is non-modifiable.

By operating the TCBE in a mode where the PCI-to-PCI bridge is disabled and the devices are hidden from the host OS, we can guarantee that the TCBE will be only path to the network and we can guarantee that the TCBE is non-by passable.

Both the TCBE driver for the host OS and the device drivers for the TCBE OS will be required. Under the assumption that the TCBE OS can provide the functionality required with high assurance, the i960 board is capable of providing hardware mechanisms needed and sufficient to support construction of a non-by-passable, non-modifiable TCBE.

D. PROTOTYPE SOFTWARE

As presented above initial hardware components of the TCBE prototype are ready and available. The glue that holds the components and makes them work as they are supposed to is the software components. Therefore, a TCBE operating system, which will initialize the TCBE and devices connected to it, will register the devices to the host system operating system and will control the operation of these devices. The TCBE operating system will reside in the ROM on the TCBE. The operating system will run as an infinitive loop and will manage the TCBE resources and devices.

The i960 board has a design similar to that of a regular PC. The board expects to start up from a specified address. It expects a branch to initialization code at first. Moreover, to initialize the board, there is minimum set of major system data structures that must be built. The structures construct an image called the initial memory image (IMI).

The TCBE operating system should perform at least initialization of the following data structures.

The initial boot record comprises the following data structures:

- Initial boot record (IBR)

IBR is the primary data structure required to initialize the i960 processor. It is a 12-word structure, which must be located at address FEFF FF30H. It consists of four components: the initial bus configuration data, the first instruction pointer, the process control block (PRCB) pointer and the bus confidence test checksum data.

- Process control block (PRCB)

The PRCB contains the base addresses for the system data structures and initial configuration information for the i960 processor. It contains the fault table base address, control table base address, interrupt table base address, system procedure table base address, and interrupt stack pointer.

- Interrupt table

The interrupt table contains pointers to the interrupt handling procedures at the associated interrupt slot.

- Fault table

The fault table contains pointers to the fault handling procedures at the associated fault slot.

- Control table

The control table is the data structure that contains the on-chip control registers values. It is automatically loaded during initialization.

Moreover, the TCBE operating system will initialize, register to the host operating system and control the operations of the NIC and the hard disk controller. So, it will require the drivers for the NIC and the hard disk controller to be put of the TCBE operating system.

E. EXPERIMENTATION

To be able to experiment with the capabilities, limitations, and vulnerabilities --if there are any-- of the i960 board, we designed some experiments. Ultimately these experiments will be among the building blocks for the TCBE operating system (O/S).

At present, the i960 board is equipped with a software monitor program that initializes the board and communicates to the development environment which can be the same or a different computer via either PCI bus or serial port. The program is called MONDB and the development environment is called MON3.30. The program initializes the board and adds communications modules to the board. In addition, the MONDB allows user programs to be written and downloaded to the board RAM. Then the downloaded code can be run on the i960 memory and the behavior of the board can be inspected, debugged.

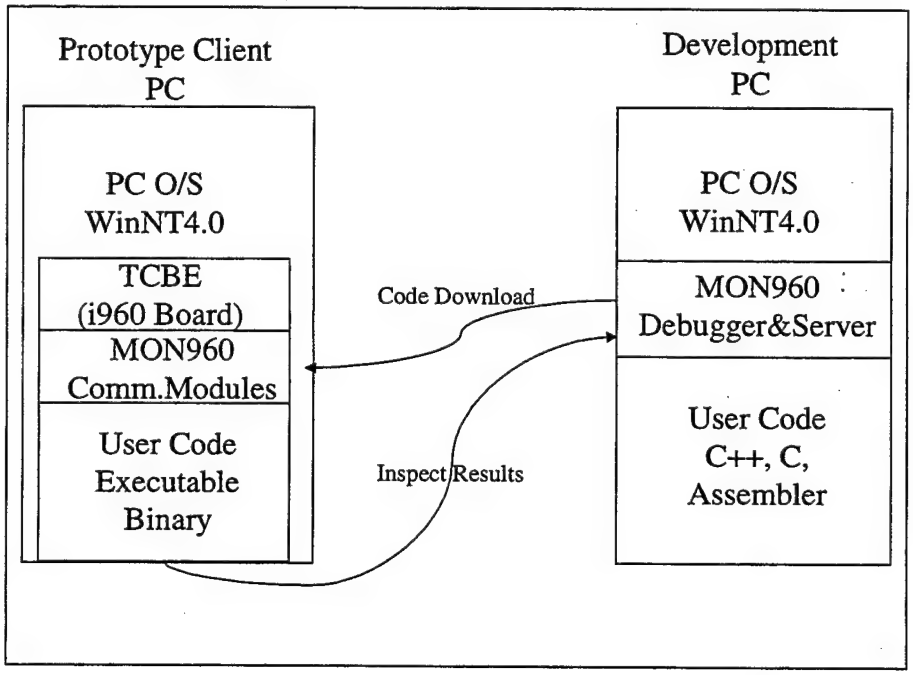

Figure 24. Development and Target Environment

We designed two experiments that will test the ability of the TCBE to control the NIC when the TCBE is working in PCI-to-PCI bridge mode.

1. Experiment 1 – TCBE's Control Of PCI-To-PCI Bridge

a. Hypothesis

The PCI-to-PCI bridge can be shutdown by configuring the registers that control the bridge.

b. Design of Experimentation

The PCI-to-PCI bridge functions exactly like a bridge that connects two roads. On one side of the bridge is the primary PCI bus, and on the other side is the secondary (TCBE's) PCI bus. The i960 board controls the operation of the PCI-to-PCI bridge via memory-mapped registers. The PCI-to-PCI bridge is created by programming two registers. One register defines where the bridge located and the other specifies the width of the bridge. The memory base register specifies the address where the bridge is formed in the host system's memory space. The memory limit register specifies the address where the bridge ends (Figure 25). By defining the bridge in the host system's memory, transactions from both sides of the bridge (primary PCI bus and TCBE PCI bus) targeted to this address range will be forwarded to the other bus.

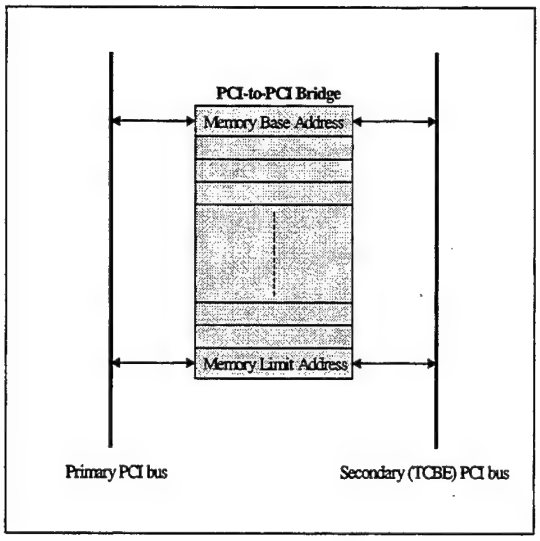

Figure 25. PCI-to-PCI bridge

One can create a bridge and define its location and how wide the bridge will be by programming these registers. One can also create a bridge by defining the base and the limit address to be the same address, which creates a bridge without any depth. It actually means that a closed bridge can be created.

c. Implementation

We will start TCBE operation as a PCI-to-PCI bridge and let the COTS operating system initialize the NIC on the TCBE PCI bus. Then we will copy the bridge's memory base address to the memory limit address. Therefore, we will shut down the bridge (Figure 26) and test if the COTS operating system can still communicate with the NIC.

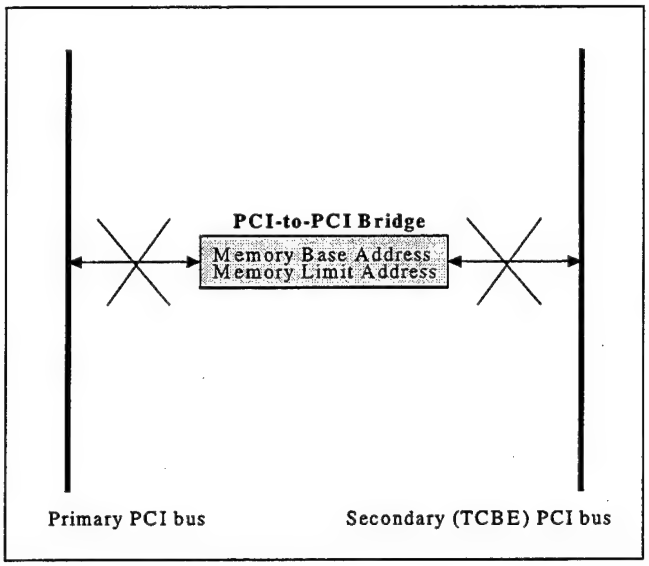

Figure 26. A Closed Bridge

Assembler code that will accomplish the task appears in Table 1.

.text	/* definitions
.globl _main	
_main:	/* main program
lda 0x00001020, g1	/*loads the memory base/limit address
lda 0x00001022, g2	/*registers addresses to g1 and g2
ld (g1), g3	/*copies the contents of base register to g3
st g3, (g2)	/*copies the contents to the limit register
forever:	
b forever	/* loops forever

Table 1. Assembler code for shutting down the bridge

d. Experimentation Data

For this experiment the memory address of the memory base register and memory limit register are required.

The internal i960 addresses for these registers are:

Memory Address Base Register	0x00001020H
Memory Address Limit Register	0x00001022H

e. Conclusion

It is possible to shut down the bridge by implementing the code above. The assembler program successfully shuts down the bridge. Therefore, we proved that the TCBE prototype will allow the devices, which are connected to the TCBE PCI bus, to be initialized, and registered to the host operating system. The TCBE holds the control of the bridge and can, at any time, start or stop the bridge function and therefore control the flow of information to or from the client PC.

2. Experiment 2- Secure Attention Key (SAK) Handling

a. Hypothesis

The SAK can be assigned to a hardware interrupt and when the SAK is pressed, the TCBE can stop the flow of information.

b. Design of Experimentation

One of the main functions of the TCBE is to provide a secure attention key (SAK) to initiate a trusted path between the user and the TCB. We want the TCBE to stop

any pending and/or continuing communications between the client PC and the network when the SAK is pressed.

The second experiment will try to stop the communications between the client PC and the network by shutting down the bridge when the SAK is pressed.

The i960 board is designed to be very similar to a regular PC. One of the primary system structures i960 board is the interrupt table. The interrupt table is a memory resident structure that contains the addresses for the interrupt handlers together with the associated interrupt vectors.

First, a brief description of how the interrupts work on the i960 board is given. The 80960 processor has dedicated interrupt pins. These pins are physically connected to the interrupt sources (devices). When an interrupt source produces an interrupt, it asserts the signal on the dedicated interrupt pin. This assertion tells the 80960 processor that there is a pending interrupt. The processor first prioritizes the pending interrupt(s) and then it starts processing one by one according to their priorities. There are 32 priorities defined, 0 being the least, 31 being the most important interrupt. Via registers, interrupts are vectored to interrupt vector numbers. When an interrupt is waiting to be processed, the processor fetches the associated vector number. Then the processor extracts the associated interrupt handler address from the interrupt table for the specified vector number. After getting the handler address, the processor jumps to the handler address and starts executing the handler from that address. When the processor finishes execution of the handler, it returns to the execution point the processor was at before the interrupt.

For the experiment, we will write an interrupt handler to shutdown the bridge to stop the data flow between the client PC and the network when a SAK interrupt occurs. The handler is the same program (Table 1) that we tested in the Experiment 1. In addition to the SAK handler, we need to load the SAK handler to an address so that we can hook an interrupt to that address by modifying the interrupt table.

c. Implementation and Experimentation Data

We can summarize the implementation of the experiment into steps:

- Write the SAK handler

We already wrote the handler to shutdown the bridge in experiment 1 (for source code see Table 1).

- Load the SAK handler to a specific address.

We chose that the handler would reside on 0xA0006000H address. Since this address was free from system code and the default handler was located close to that address. The assembler code is shown in Table 2.

- Select an external interrupt

As a test, we chose the external interrupt number 4 (INT#4) because INT#4 can be simulated by setting a bit in the interrupt pending register (IPND) at memory address FF008500H. If it works using the interrupt number four (INT#4), we will replace the INT#4 with the non-maskable interrupt (NMI#), which we cannot simulate. For the NMI# case we must physically connect an interrupt source to the NMI# pin of the 80960 processor.

.text	/*declarations
.globl _main	
_main:	/* main program
lda SAK_handler, g1	/*loads the handler address to g1
lda SAK_handler_end, g3	/*loads the address of
init_copy :	/* program piece that copies the handler to specified address A0006000H
ldq (g1), r4	/*copies the contents SAK handler to r4
addo 16, g1, g1	/*increments the destination address
stq r4, (g2)	/*moves it to the new address space
add 16, g2, g2	/*increments the target address
cmpobl g1, g3, init_copy	/*checks if it is end of handler

Table 2. Handler copying program

- Map the selected interrupt to a interrupt vector number as shown in Table 3

For INT#4, we chose interrupt vector #18. We programmed the board to map INT#4 to vector#18 by placing 0x00000001H into interrupt mapping register number 1 at the memory address FF008524H. In the NMI# case, it is not necessary to map the interrupt to a vector since the NMI# vector number is fixed at vector #248.

lda 0xff008510, g4	/*set the ICON register
ldconst 0x00004C00, g6	
st g6, (g4)	
lda 0xff008524, g4	/*set vector#18 to INT#4
ldconst 0x00000001, g6	
st g6, (g4)	

Table 3. Mapping of INT#4 to vector #18

- Modify the interrupt table and insert the handler into the associated vector slot

Interrupt vector #18 corresponds to the offset address 4CH in the interrupt table, so we modified the interrupt table and inserted the handler address (0xA0006000) into the address 0xA000924CH which is the sum of base interrupt table address plus offset-0xA0009200H+0x0000004CH). For NMI#, the offset is fixed and is equal to 3E4H, so the modification should be placed into the address 0xA00095E4H. The assembly code appears in Table 4.

lda 0xA0006000, g2	/*load the handler address to g2
lda 0xA000924C, g5	/*load the interrupt table slot to g5
st g2, (g5)	/*insert the new handler address to int. table

Table 4. Modifying interrupt table

- Create an interrupt

For the INT#4, it is possible to simulate an interrupt by setting the fifth bit within the interrupt pending register (IPND) at memory address ff008500h. Therefore, an external interrupt can be simulated and the SAK handler can be tested. For NMI#, we actually need to connect an interrupt source such as a button. The assembler code that simulates an INT#4 interrupt appears in Table 5.

```
lda 0xFF008500, g4          /*load address of IPND register
ldconst 0x00000001, g6      /*set the INT#4 as if there is an interrupt
st g6, (g4)
```

Table 5. Simulating interrupt

d. Conclusion

We proved that it is possible to assign an external interrupt to the SAK. Moreover, we showed that we can process that interrupt as a SAK interrupt and cause the bridge to shut down by stimulating an interrupt handler.

F. CONCLUSION

Some preliminary TCBE functionallity has been constructed. Although, it is very primitive, it demonstrates that we are able to make it work and proved that it is possible to run TCBE in mode 0. Running TCBE in mode 0 means that the TCBE devices are declared as public to the COTS O/S. Nevertheless, the TCBE can still control the data flow to or

from the network and revoke access to the devices from the O/S and halt the data flow to and from the network when the SAK is pressed.

The ability to operate in mode 0 allows the client PC to be used as a regular desktop workstation with a standard NIC. But, when the SAK is pressed the TCBE can disconnect the client from the network and perform its communications setup functions and if the initial setup is completed successfully then the client PC can be reset and the TCBE can start running in mode 3 which will hide the devices from the client PC.

V. CONCLUSION

In the thesis, the multi level secure local area network (MLS LAN) problem is introduced. A solution to this problem is a local area network consisting of a high assurance server and client PCs equipped with custom boards. The custom boards are called the TCBE. Each TCBE is a PCI board with a processor. The TCBE modifies the PC architecture without modifying the motherboard. It actually extends the PCI bus and reroutes the network interface card and the hard disk controller from the primary PCI bus to the TCBE PCI bus. Therefore, the TCBE controls the operations and the availability of the devices from the host PC operating system. Functionally, the TCBE extends the TCB to the client PC, generates a direct path between the user and the TCB and supports the high assurance server's security policy.

In this thesis, Chapter I introduced the historical background of computer security and described the problem of securely sharing information at different classification levels. Then, the basic computer security concepts related to the problem were discussed. Finally, the first chapter proposed a solution to the problem.

Chapter II analyzed the requirements for the TCBE. In addition, the requirements to support a distributed trusted path is presented. In the second chapter, we also investigated the design and implementation issues related to the TCBE. We specified the location of the TCBE within the client PC and defined the peripherals required on the TCBE. Therefore, we sketched the TCBE in terms of its location on the client PC and the functional units that it will possess.

After deciding upon the location of the TCBE, in Chapter III, we analyzed the interface technologies and supplied technical information for each interface and related the TCBE to these interfaces and techologies.

In the Chapter IV, we suggested a TCBE prototyping environment. We designed two experiments to prove that the TCBE can control the rerouted devices on the TCBE PCI bus and either provide the devices for the COTS O/S use or hide the devices from the COTS O/S. Moreover, we tested and proved right at the end of the experimentation.

A. RECOMMENDATIONS FOR FUTURE WORK

1. The TCBE Hard Disk Issues

One of the requirements for the TCBE is to control the object reuse in the client PC. For object reuse control, the TCBE should essentially control both system main memory and the hard disk(s).

For the preliminary TCBE prototype, hard disk controller would be implemented by removing the PC's hard drive(s) and installing a PCI SCSI controller into theTCBE PCI bus and connecting one or more SCSI hard drive(s). These drive(s) will serve a system wide permanent storage. We actually tested this setup, however, we were not successful. The prototype client PC operating system (OS) was Windows NT and when we relocated the SCSI controller, Windows NT was not able to recognize the SCSI hard disk on the TCBE PCI bus. When the drives and the controller were installed to the client PC's primary PCI bus, Windows NT OS recognized and initialized the hard drives

Therefore, investigation of this behavior is needed to determine whether it is a limitation of the client OS (Windows NT) or a configuration issue that can easily be solved.

Another issue to be solved in the preceeding work is zeroization of main memory after each session. Agacayak [Ref. 8] has done some experiments regarding these aspects of the project. Further analysis is needed to determine if there is a possibility of initiating a cleaning process for system memory or a possibility of manipulation of refresh rates for the main memory to an unacceptible level. By manipulating the refresh rates to an abnormal value that one can force the memory to lose its contents, therefore allowing the memory to be cleaned from the TCBE via PCI configuration cycles.

The last issue to be considered is that of a driver for the hard disk. Since the client OS has a driver for the hard disk input and output (I/O) operations, it needs to be replaced with a TCBE driver that will recognize and route the hard disk I/O data to the TCBE so that the TCBE will process the data and perform hard disk I/O.

2. The TCBE Display Issues

The TCBE display and its interface are specified in Chapter III, but the preliminary TCBE prototype never included a display or an interface. Therefore, the display and interface should be constructed and interfaced to the TCBE.

In addition to physically constructing these elements, a TCBE display driver that will run under the TCBE OS should be written

3. The TCBE Keyboard Issues

The TCBE keyboard was also specified in the Chapter III. The preliminary TCBE prototype did not include a keyboard. Therefore, a physical keyboard and its interface have to be constructed and integrated to the TCBE board.

A driver that will handle keyboard operations, and an interrupt routine that will keep track of keyboard entries and process the keyboard operations needs to be written and integrated into the TCBE OS.

4. The TCBE Operating System and The TCBE Driver

In Chapter IV, it is described that the TCBE will operate in two different modes: PCI-to-PCI bridge mode and Gatekeeper mode.

For each mode of operation, there are unique functions that need to be performed as well as some common operations that need to be performed in both modes. For example SAK handling functions should be invokable at all times, but port remapping should be initiated when the TCBE is in gatekeeper mode. These functions should be added and contribute to the TCBE OS.

For the PCI-to-PCI bridge operation, the TCBE should be initialized at least to a PCI-to-PCI bridge mode and if additonal configuration is needed it should be handled at TCBE initialization. In this mode, the TCBE allows the client OS to probe and configure the bridge and the devices residing on the PCI bus, which means that the client OS has control of these devices. But, when the secure attention key (SAK) is pressed, the TCBE

needs to gain the control of these devices. At this stage, the TCBE has to disable the ability of the client PC to configure the bridge and the devices to guarantee that the OS does not regain the control of the devices. In addition, the TCBE needs to establish a communication channel which includes a trusted path.

Up to this point, the TCBE does not need to present a special driver to the client OS, because it just operates as a PCI-to-PCI bridge. On the other hand, the TCBE OS is only required to perform some basic operations such as initialization, keyboard and display driving, communication channel establishment and SAK handling.

When the handshake between the high assurance server (HAS) and the TCBE has been performed successfully, the TCBE needs to reinitialize itself into its other operating mode: that of a gatekeeper. At the same time, the TCBE should force the client PC to reboot. In this mode, the TCBE will completely hide the devices on its PCI bus from the client OS and initialize them internally. Therefore the TCBE OS should include additional routines that will initialize and drive these devices through the operation of the TCBE. Moreover, the functions such as encryption, data checksumming, IPSEC operations that are required during the operation of the TCBE, should be added to the TCBE OS.

Another issue in this operation is that the TCBE will operate differently from a basic bridge. It will take over the functions of a network interface and hard disk. The TCBE will handle network and data I/O packets. Therfore, the TCBE should at least register itself to the client OS as a hard disk driver, and network interface card driver. A special TCBE driver needs to be written for handling hard disk controller, and network interface card functions.

B. SUMMARY

This thesis proved that the TCBE could provide a solution to the MLS LAN problem. It also provided a prototyping environment for primitive TCBE experiments. A set of recommendations for future work has been specified.

The i960 board proved to the feasible platform to provide hardware support for a TCBE. However, for the most part, a full functional TCBE prototype will be an effort of programming the prototype board. Therefore, a complete understanding of the prototype development board and the programming environment will be crucial to the success of the prototype.

APPENDIX A. PROGRAMMING I960 BOARD

All the informationpresented in section A and section B can be found on [Ref. 19][Ref. 20][Ref. 21]. These sections are included in the thesis to give programmers a brief insight and guidance.

A. HOW TO WRITE ASSEMBLER PROGRAMS FOR I960 RM BOARD

The 80960 processor is an Intel RISC processor. It has its own Assembler language [Ref. 18]. An i960 assembler program can be written in any text editor. However, an assembler program should be saved as .s file. For example, assembler1.s.

There is a basic syntax that has to be followed while writing a program. A program must have a main routine, and this routine should be declared global. The comments can be added to a program but they should start with /*. Generally, a program completes with an infinitive loop so that it will not spill to the other programs.

Here is a program example (example1.s):

```
.text
.globl _main            /*declaration of main
_main:                  /* main program
lda SAK_handler, g1     /*loads the handler address to g1
lda SAK_handler_end, g3 /*loads the address of
_forever:               /*infinitive loop
b forever
```

B. HOW TO COMPILE AND LINK THE PROGRAMS FOR I960 RM BOARD

To be able to download, run and debug a program into the i960 board, the program must be compiled and linked into an executable file. The compiler compiles the assembler files into object files. And the linker links the object file with the required libraries and produces an executable file. The process of linking and compiling programs and the options of compiling and linking have been presented in [Ref. 20] and [Ref. 21]. The compiler and the linker is started and controlled by a command line.

However, I will show here a practical and most useful options for compiling and linking an assembler file:

- Compiling

After the assembler program is written and saved as .s file. We will go to the command line and we will type:

gas960c –ARM –V assembler1.s

This line will control and compile assembler.s to assembler.o file. If there is any errors in the assembler.s file, the compiler will prompt the error and quit without compilation. The options for the command are:

-ARM	shows the processor type for compilation (RM)
-V	displays version information

- Linking

Again we will start a command shell and type:

gld960 –ARM –Fcoff –Tcytx –o assembler1.bin assembler.o

This line links the assembler.o object file with cytx libraries and generates an executable file named assembler.bin. The options for the command are:

-ARM	shows the processor type for compilation (RM)
-v	displays version information
-o	generates an executable
-Fcoff	type of output file (coff object module format)
-Tcytx	libraries to be used while compiling

THIS PAGE INTENTIONALLY LEFT BLANK.

LIST OF REFERENCES

1. Sterne, D. F., "On The Buzzword "Security Policy"," *Proceedings of the IEEE Computer Society Symposium on Security and Privacy*, Oakland, CA, 1991.

2. Anderson, J. P., *Computer Security Technology Planning Study*, ESD-TR-73-51, Volume 1, Hanscom AFB, MA, 1972 (also available DTIC AD-758206).

3. Schell, R., "Computer Security: The Achilles' Heel of the Electronic Air Force", *Air University Review*, January 1978.

4. *Department of Defense Trusted Computer System Evaluation Criteria*, DoD 5200.28-STD, National Computer Security Center, December 1985.

5. *Evaluation Criteria for Information Technology Security*, ISO/IEC JTC 1/SC27 N 2161, ISO/IEC 15408:1999 (E), 18 December 1998.

6. Balmer, S. *Framework For A High Assurance Security Extension to Commercial Network Clients,* Master's Thesis, Naval Postgraduate School, Monterey, CA, September 1999.

7. Saltzer, J. H., Schroder, M. D., "The Protection of Information in Computer Systems", *Proceedings of the IEEE, Vol. 63, No. 9*, September 1975.

8. Agacayak, Cihan. *The Object Reuse Control on the Client,* Master's Thesis, Naval Postgraduate School, Monterey, CA, March 2000.

9. Messner, H. P., *The Indispensable PC Hardware Book,* Second Edition, Addison-Wesley Pub. Co., Essex, England, 1995.

10. Shanley, T., Anderson, D., *PCI System Architecture,* Third Edition, Addison-Wesley Pub. Co., Reading, MA, 1995.

11. Bryer-Joyner, S., Heller S. *Secure Local Area Network Services for a High-Assurance Multilevel Network*, Naval Postgraduate School, Master's Thesis, Monterey, CA, March 1999.

12. Stallings, W., *Data And Computer Communications,* Fifth Edition, Prentice Hall, Upper Saddle River, NJ, 1996.

13. *3C90X and 3C90XB NICs Technical Reference,* Part Number 89-0766-000, 3COM, 1998.

14. *i960 RM/RN I/O Processor Developer's Manual,* Order Number 273158-001, Intel Corporation, 1998.

15. Zecker, C., *TCP/IP Administration,* The IDG Books Worldwide Inc., Foster City, CA, 1997.

16. Tanenbaum, A., *Computer Networks,* Third Edition, Prentice Hall, Upper Saddle River, NJ, 1996.

17. *i960 RM/RN Evaluation Platform Board Manual,* Order Number 273160-003, Intel Corporation, 1998.

18. *i960 Processor Assembler User's Guide,* Order Number 485276-007, Revision 6.5, Intel Corporation, 1998.

19. *Mon960 Debug Monitor User's Guide,* Order Number 484290-008, Revision 3.3, Intel Corporation, 1998.

20. *GDB960 User's Guide,* Order Number 485546-005, Revision 6.5, Intel Corporation, 1998.

21. *i960 Processor Compiler User's Guide,* Order Number 651230-004, Revision 6.5, Intel Corporation, 1998.

INITIAL DISTRIBUTION LIST

No. Copies

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., Ste 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Chairman, Code CS 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943-5193

4. Chairman, Code EC 1
   Electrical Engineering Department
   Naval Postgraduate School
   Monterey, CA 93943-5121

4. Dr. Cynthia E. Irvine 3
   Computer Science Department Code CS/Ic
   Naval Postgraduate School
   Monterey, CA 93943-5193

5. Mr. James P. Anderson 1
   James P. Anderson Company
   Box 42
   Fort Washington, PA 19034

6. Dr.William A.Arbaugh 1
   WAA Associates, LLC.
   4264 Hermitage Dr.
   Ellicott City, MD. 21042

7. Mr. Paul Pitelli 1
   National Security Agency
   Research and Development Building
   R2, Technical Director

9800 Savage Road
Fort Meade, MD 20755-6000

8. Dr. Lee Taylor 1
National Security Agency
Research and Development Building
R22, Chief
9800 Savage Road
Fort Meade, MD 20755-6000

9. Ms. Donna Belt 1
National Security Agency
Research and Development Building
R23, Chief
9800 Savage Road
Fort Meade, MD 20755-6000

10. CAPT Dan Galik 1
Space and Naval Warfare Systems Command
PMW 161
Building OT-1, Room 1024
4301 Pacific Highway
San Diego, CA 92110-3127

11. Commander, Naval Security Group Command 1
Naval Security Group Headquarters
9800 Savage Road
Suite 6585
Fort Meade, MD 20755-6585

12. Ms. Deborah M. Cooper 1
Deborah M. Cooper Company
P. O. Box 17753
Arlington, VA 22216

13. Ms. Louise Davidson 1
N643
Presidential Tower 1
2511 South Jefferson Davis Highway
Arlington VA 22202

14. Mr. William Dawson.....1
Community CIO Office
Washington DC 20505

15. Deniz Kuvvetleri Komutanligi.....1
Personel Tedarik ve Yetistirme Daire Baskanligi
06100 Bakanliklar, Ankara
TURKEY

16. Bora Turan.....2
Deniz Harp Okulu Komutanligi
Elektronik Ogretim Uyesi
81704 Tuzla, Istanbul
TURKEY

17. Deniz Harp Okulu Komutanligi.....2
Kutuphane
81704 Tuzla, Istanbul
TURKEY